Package ‘Rbeast’

May 18, 2022

Type Package
Version 0.9.4
Date 2022-05-17
Title Bayesian Change-Point Detection and Time Series Decomposition
Author Kaiguang Zhao [aut, cre],
Tongxi Hu [aut],
Yang Li [aut],
Jack Dongarra [ctb],
Cleve Moler [ctb]
Maintainer Kaiguang Zhao <zhao.1423@osu.edu>
Depends R (>= 2.10.0), methods, utils
Description Interpretation of time series data is affected by model choices. Different models can give different or even contradicting estimates of patterns, trends, and mechanisms for the same data—a limitation alleviated by the Bayesian estimator of abrupt change, seasonality, and trend (BEAST) of this package. BEAST seeks to improve time series decomposition by forgoing the "single-best-model" concept and embracing all competing models into the inference via a Bayesian model averaging scheme. It is a flexible tool to uncover abrupt changes (i.e., change-points), cyclic variations (e.g., seasonality), and nonlinear trends in time-series observations. BEAST not just tells when changes occur but also quantifies how likely the detected changes are true. It detects not just piecewise linear trends but also arbitrary nonlinear trends. BEAST is applicable to real-valued time series data of all kinds, be it for remote sensing, economics, climate sciences, ecology, and hydrology. Example applications include its use to identify regime shifts in ecological data, map forest disturbance and land degradation from satellite imagery, detect market trends in economic data, pinpoint anomaly and extreme events in climate data, and unravel system dynamics in biological data. Details on BEAST are reported in Zhao et al. (2019) <doi:10.1016/j.rse.2019.04.034>.

LazyData true
Imports grid
License GPL (>= 2)
URL https://github.com/zhaokg/Rbeast
NeedsCompilation yes
beast

Bayesian time series decomposition for changepoint, trend, and periodicity or seasonality

Description

A Bayesian model averaging algorithm called BEAST to decompose time series or 1D sequential data into individual components, such as abrupt changes, trends, and periodic/seasonal variations. BEAST is useful for changepoint detection (e.g., breakpoints or structural breaks), nonlinear trend analysis, time series decomposition, and time series segmentation.

Usage

```
beast(
  y, 
  start = 1, 
  deltat = 1, 
  season = c('harmonic','dummy','svd','none'), 
  freq = NA, 
  scp.minmax = c(0,10), 
  sorder.minmax = c(0,5), 
  sseg.min = NULL, 
  tcp.minmax = c(0,10), 
  torder.minmax = c(0,1), 
  tseg.min = NULL, 
  detrend = FALSE, 
  deseasonalize= FALSE, 
)
mcmc.seed = 0,
mcmc.burnin = 200,
mcmc.chains = 3,
mcmc.thin = 5,
mcmc.samples = 8000,
ci = FALSE,
precValue = 1.5,
precPriorType = \{'componentwise', 'uniform', 'constant', 'orderwise'\},
print.options = TRUE,
print.progress = TRUE,
gui = FALSE,
...

Arguments

\(y\) a vector for an evenly-spaced regular time series. Missing values such as NA and NaN are allowed.

- If \(y\) is irregular or unordered in time (e.g., multiple years of daily data spanning across leap years: 365 points in some years, and 366 in others), use the \texttt{beast.irreg} function instead.
- If \(y\) is a matrix or 3D array consisting of multiple regular or irregular time series (e.g., stacked images), use \texttt{beast123} instead.
- If \(y\) is an object of class 'ts', its time attributes (i.e., start, end, frequency) will be used to specify the next several args such as start, delta, freq, and season: No need to provide them explicitly; even if provided, the values are ignored to honor the time attributes of \(y\). For example, if \(y\) has a frequency = 1, season = 'none' is always assumed; if \(y\) has a frequency > 1 (i.e., with a periodic component) but season='none' is specified by the user, 'none' will be replaced by 'harmonic'.

If a list of multiple time series is provided for \(y\), the multivariate version of the BEAST algorithm will be invoked to decompose the multiple time series and detect common changepoints altogether. This feature is experimental only and under further development. Check \texttt{ohio} for a working example.

\(start\) numeric (default to 1.0) or Date; the time of the 1st datapoint of \(y\). It can be specified as a scalar (e.g., 2021.0644), a vector of three values in the order of Year, Month, and Day (e.g., c(2021,1,24)), or a R’s Date object (e.g., as.Date('2021-1-24')).

\(deltat\) numeric (default to 1.0); the time interval between consecutive data points. Its unit should be consistent with \(start\). If \(start\) takes a numeric scalar, the unit is arbitrary and irrelevant to beast (e.g., 2021.3 can be of any unit). If \(start\) is a vector of Year, Month, and Day or an R’s Date, \(deltat\) has the unit of YEAR. For example, if \(start\)=c(2021,1,24) for a monthly time series, \(start\) is converted to a fractional year 2021+(24-0.5)/365=2021.0644 and \(deltat\)=1/12 needs to be set in order to specify the monthly interval.

\(season\) characters (default to 'harmonic'); specify if \(y\) has a periodic component or not. Four strings are possible.
• 'none': y is trend-only; no periodic components are present in the time series. The args for the seasonal component (i.e., sorder.minmax, scp.minmax and sseg.max) will be irrelevant and ignored.

• 'harmonic': y has a periodic/seasonal component. The term season is a misnomer, being used here to broadly refer to any periodic variations present in y. The periodicity is NOT a model parameter estimated by BEAST but a known constant given by the user through freq. By default, the periodic component is modeled as a harmonic curve—a combination of sines and cosines.

• If 'dummy', the same as 'harmonic' except that the periodic/seasonal component is modeled as a non-parametric curve. The harmonic order arg sorder.minmax is irrelevant and is ignored.

• If 'svd', the same as 'harmonic' except that the periodic/seasonal component is modeled as a linear combination of function bases derived from a Single-value decomposition. The SVD-based basis functions are more parsimonious than the harmonic sin/cos bases in parameterizing the seasonal variations; therefore, more subtle changepoints are likely to be detected.

freq numeric. Needed only for data with a periodic/cyclic component (i.e., season='harmonic' or 'dummy') and ignored for trend-only data (i.e., season='none'). It specifies the number of samples/datapoints per cycle (e.g., a time series of monthly observations with an annual period has a frequency of 12); it may be a decimal real number (e.g., a time series of bi-hourly observations with a period of 37.5 hrs has a freq of 37.5/2=18.75). The period of the cyclic component in the unit of time is period=deltat*freq. freq is not a model parameter of BEAST and has to be specified by the user, but if freq is missing, BEAST first attempts to guess its value via auto-correlation before fitting the model: in this case, freq is assumed to be an integer.

sorder.minmax a vector of 2 integers (>=1); the min and max harmonic orders considered to fit the seasonal component. sorder.minmax is used only if the time series has a seasonal component (i.e., season='harmonic') and ignored for trend-only data or when season='dummy'. If the min and max orders are equal (sorder.minmax[1]=sorder.minmax[2]), BEAST assumes a constant harmonic order used and won’t infer the posterior probability of harmonic orders.

scp.minmax a vector of 2 integers (>=0); the min and max number of seasonal changepoints (scp) allowed in segmenting the seasonal component. scp.minmax is used only if y has a seasonal component (i.e., season='harmonic' or 'dummy') and ignored for trend-only data. If the min and max changepoint numbers are equal, BEAST assumes a constant number of scp and won’t infer the posterior probability of the number of changepoints, but it still estimates the occurrence probability of the changepoints over time (i.e., the most likely times at which these changepoints occur). If both the min and max numbers are set to 0, no changepoints are allowed; then a global harmonic model is used to fit the seasonal component, but still, the most likely harmonic order will be inferred if sorder.minmax[1] is not equal to sorder.minmax[2].

sseg.min an integer (>0); the min segment length allowed between two neighboring season changepoints. That is, when fitting a piecewise harmonic seasonal model, no two changepoints are allowed to occur within a time window of length sseg.min.
sseg.min must be an unitless integer—\textit{the number of time intervals/data points so that the time window in the original unit is sseg.min*deltat}. sseg.min defaults to NULL and its value will be given a default value in reference to \textit{freq}.

torder.minmax

A vector of 2 integers (\textit{\textgreater}=0); the min and max orders of the polynomials considered to fit the trend component. The 0-th order corresponds to a constant term/a flat line and the 1st order is a line. If torder.minmax[1]=torder.minmax[2], BEAST assumes a constant polynomial order used and won’t infer the posterior probability of polynomial orders.

tcp.minmax

A vector of 2 integers; the min and max number of trend change points (tcp) allowed in segmenting the trend component. If the min and max change point numbers are equal, BEAST assumes a constant number of change points and won’t infer the posterior probability of the number of change points for the trend, but it still estimates the occurrence probability of the change points over time (i.e., the most likely times at which these change points occur in the trend). If both the min and max numbers are set to 0, no change points are allowed; then a global polynomial trend is used to fit the trend component, but still, the most likely polynomial order will be inferred if torder.minmax[1] is not equal to torder.minmax[2].

tseg.min

An integer (\textit{\textgreater}=0); \textit{the min segment length allowed between two neighboring trend change points}. That is, when fitting a piecewise polynomial trend model, no two change points are allowed to occur within a time window of length tseg.min. tseg.min must be an unitless integer—\textit{the number of time intervals/data points so that the time window in the original unit is tseg.min*deltat}. tseg.min defaults to NULL and its value will be given a default value in reference to \textit{freq} if the time series has a cyclic component.

detrend

\textit{logical}; If \textbf{TRUE}, a global trend is first fitted and removed from the time series before running BEAST; after BEAST finishes, the global trend is added back to the BEAST result.

deseasonalize

\textit{logical}; If \textbf{TRUE}, a global seasonal model is first fitted and removed from the time series before running BEAST; after BEAST finishes, the global seasonal curve is added back to the BEAST result. deseasonalize is ignored if season='none' (i.e., trend-only data).

mcmc.seed

\textit{integer (\textgreater}=0); the seed for the random number generator used for Monte Carlo Markov Chain (mcmc). If mcmc.seed=0, an arbitrary seed is picked and the fitting results vary across runs. If fixed to the same non-zero integer, the result can be re-produced for different runs. But the results from the same seed may still vary if run on different computers because the random generator library depends on CPU’s instruction sets.

mcmc.chains

\textit{integer (\textgreater}=0); the number of MCMC chains.

mcmc.thin

\textit{integer (\textgreater}=0); a factor to thin chains (e.g., if thinningFactor=5, samples will be taken every 3 iterations)

mcmc.burnin

\textit{integer (\textgreater}=0); the number of burn-in samples discarded at the start of each chain

mcmc.samples

\textit{integer (\textgreater}=0); the number of samples collected per MCMC chain. The total number of iterations is (burnin+samples*thin)*chains.

\textit{ci}

\textit{boolean}; If \textbf{TRUE}, credible intervals (i.e., out$season$CI or out$trend$CI) will be computed for the estimated seasonal and trend components. Computing CI
is time-consuming, due to sorting, so set ci to FALSE if a symmetric credible interval (i.e., out$trend$SD and out$season$SD) suffices.

**precValue**
numeric (>0); the hyperparameter of the precision prior; the default value is 1.5. precValue is useful only when precPriorType='constant', as further explained below.

**precPriorType**
characters. It takes one of 'constant', 'uniform', 'componentwise' (the default), and 'orderwise'. Below are the differences between them.

1. 'constant': the precision parameter used to parameterize the model coefficients is fixed to a constant specified by precValue. In other words, precValue is a user-defined hyperparameter and the fitting result may be sensitive to the chosen values of precValue.

2. 'uniform': the precision parameter used to parameterize the model coefficients is a random variable; its initial value is specified by precValue. In other words, precValue will be inferred by the MCMC, so the fitting result will be insensitive to the choice in precValue.

3. 'componentwise': multiple precision parameters are used to parameterize the model coefficients for individual components (e.g., one for season and another for trend); their initial values is specified by precValue. In other words, precValue will be inferred by the MCMC, so the fitting result will be insensitive to the choice in precValue.

4. 'orderwise': multiple precision parameters are used to parameterize the model coefficients not just for individual components but also for individual orders of each component; their initial values is specified by precValue. In other words, precValue will be inferred by the MCMC, so the fitting result will be insensitive to the choice in precValue.

**print.options**
boolean. If TRUE, the full list of input parameters to BEAST will be printed out prior to the MCMC inference; the naming for this list (e.g., metadata, prior, and mcmc) differs slightly from the input to beast, but there is a one-to-one correspondence (e.g., prior$trendMinSepDist=tseg.min). Internally, beast converts the input parameters to the forms of metadata, prior, and mcmc. Type 'View(beast)' to see the details or check the beast123 function.

**print.progress**
boolean; If TRUE, a progressbar will be displayed.

**gui**
boolean. If TRUE, BEAST will be run with a GUI window to show an animation of the MCMC sampling in the model space step by step; as an experimental feature, "gui=TRUE" works only for Windows x64 systems not Windows 32 or Linux/Mac.

... additional parameters. There are many more settings for the implementation but not made available in the beast() interface; please use the function beast123() instead.

**Value**
The output is an object of class "beast". It is a list, consisting of the following variables. Its structure is the same as the outputs from the other two alternative functions beast.irreg and beast123. In the explanations below, we assume the input y is a single time series of length N:
time: a vector of size 1xN: the times at the N sampled locations. By default, it is simply set to 1:N

data: a vector, matrix, or 3D array; this is a copy of the input y if extra$dumpInputData = TRUE. If extra$dumpInputData=FALSE, it is set to NULL. If the original input y is irregular (as in beast.irreg), the copy here is the regular version aggregated from the original at the time interval specified by deltat (in beast.irreg or metadata$deltaTime (in beast123)).

marg_lik: numeric; the average of the model marginal likelihood; the larger marg_lik, the better the fitting for a given time series.

R2: numeric; the R-square of the model fitting.

RMSE: numeric; the RMSE of the model fitting.

sig2: numeric; the estimated variance of the model error.

trend: a list object consisting of various outputs related to the estimated trend component:

- ncp: [Number of ChangePoints]. a numeric scalar; the mean number of trend changepoints. Individual models sampled by BEAST has a varying dimension (e.g., number of changepoints or knots), so several alternative statistics (e.g., ncp_mode, ncp_median, and ncp_pct90) are also given to summarize the number of changepoints. For example, if ncnc$(samples=10, the numbers of changepoints for the 10 sampled models are assumed to be c(0, 2, 4, 1, 1, 2, 7, 6, 6, 1). The mean ncp is 3.1 (rounded to 3), the median is 2.5 (2), the mode is 1, and the 90th percentile (ncp_pct90) is 6.5.

- ncp_mode: [Number of ChangePoints]. a numeric scalar; the mode for number of changepoints. See the above for explanations.

- ncp_median: [Number of ChangePoints]. a numeric scalar; the median for number of changepoints. See the above for explanations.

- ncp_pct90: [Number of ChangePoints]. a numeric scalar; the 90th percentile for number of changepoints. See the above for explanations.

- ncpPr: [Probability of the Number of ChangePoints]. A vector of length (tcp.minmax[2]+1)=tcp.max+1. It gives a probability distribution of having a certain number of trend changepoints over the range of [0,tcp.max]; for example, ncpPr[1] is the probability of having no trend changepoint; ncpPr[i] is the probability of having (i-1) changepoints: Note that it is ncpPr[i] not ncpPr[i-1] because ncpPr[1] is used for having zero changepoint.

- cpOccPr: [ChangePoint OCCurrence PRobability]. a vector of length N; it gives a probability distribution of having a changepoint in the trend at each point of time. Plotting cpOccPr will depict a continous curve of probability-of-being-changepoint. Of particular note, in the curve, a higher peak indicates a higher chance of being a changepoint only at that particular SINGLE point in time and does not necessarily mean a higher chance of observing a changepoint AROUND that time. For example, a window of cpOccPr values c(0,0,0.5,0,0) (i.e., the peak prob is 0.5 and the summed prob is 0.5) is less likely to be a changepoint compared to another window c(0.1,0.2,0.21,0.2,0.1) (i.e., the peak prob is 0.21 but the summed prob is 0.71).
• order: a vector of length N; the average polynomial order needed to approximate the fitted trend. As an average over many sampled individual piece-wise polynomial trends, order is not necessarily an integer.

• cp: [Changepoints] a vector of length tcp.max=tcp.minmax[2]; the most possible changepoint locations in the trend component. The locations are obtained by first applying a sum-filtering to the cpOccPr curve with a filter window size of tseg.min and then picking up to a total prior$MaxKnotNum/tcp.max of the highest peaks in the filtered curve. NaNs are possible if no enough changepoints are identified. cp records all the possible changepoints identified and many of them are bound to be false positives. Do not blindly treat all of them as actual changepoints.

• cpPr: [Changepoints PRobability] a vector of length tcp.max=tcp.minmax[2]; the probabilities associated with the changepoints cp. Filled with NaNs for the remaining elements if ncp<tcp.max.

• cpCI: [Changepoints Credible Interval] a matrix of dimension tcp.max x 2; the credible intervals for the detected changepoints cp.

• cpAbruptChange: [Abrupt change at Changepoints] a vector of length tcp.max; the jumps in the fitted trend curves at the detected changepoints cp.

• Y: a vector of length N; the estimated trend component. It is the Bayesian model averaging of all the individual sampled trend.

• SD: [Standard Deviation] a vector of length N; the estimated standard deviation of the estimated trend component.

• CI: [Standard Deviation] a matrix of dimension N x 2; the estimated credible interval of the estimated trend. One vector of the matrix is for the upper envelope and another for the lower envelope.

• slp: [Slope] a vector of length N; the time-varying slope of the fitted trend component.

• slpSD: [Standar Deviation of Slope] a vector of length N; the SD of the slope for the trend component.

• slpSgnPosPr: [PRobability of slope having a positive sign] a vector of length N; the probability of the slope being positive (i.e., increasing trend) for the trend component. For example, if slpSgnPosPr=0.80 at a given point in time, it means that 80% of the individual trend models sampled in the MCMC chain has a positive slope at that point.

• slpSgnZeroPr: [PRobability of slope being zero] a vector of length N; the probability of the slope being zero (i.e., a flat constant line) for the trend component. For example, if slpSgnZeroPr=0.10 at a given point in time, it means that 10% of the individual trend models sampled in the MCMC chain has a zero slope at that point. The probability of slope being negative can be obtained from 1-slpSgnZeroPr-slpSgnPosPr.

• pos_ncp:

• neg_ncp:

• pos_ncpPr:

• neg_ncpPr:

• pos_cpOccPr:

• neg_cpOccPr:
• pos_cp
• neg_cp
• pos_cpPr
• neg_cpPr
• pos_cpAbruptChange
• neg_cpAbruptChange
• pos_cpCI
• neg_cpCI: The above variables have the same outputs as those variables without the prefix 'pos' and 'neg', except that we differentiate the changepoints with a POSitive jump in the trend from those changepoints with a NEGative jump. For example, pos_ncp refers to the average number of trend changepoints that jump up (i.e., positively) in the trend.
• inc_ncp
• dec_ncp
• inc_ncpPr
• dec_ncpPr
• inc_cpOccPr
• dec_cpOccPr
• inc_cp
• dec_cp
• inc_cpPr
• dec_cpPr
• inc_cpAbruptChange
• dec_cpAbruptChange
• inc_cpCI
• dec_cpCI: The above variables have the same outputs as those variables without the prefix 'inc' and 'dec', except that we differentiate the changepoints at which the trend slope increases from those changepoints at which the trend slope decreases. For example, if the trend slopes before and after a chngpt is 0.4 and 2.5, then the changepoint is counted toward inc_ncp.

season a list object consisting of various outputs related to the estimated seasonal/periodic component:
• ncp: [Number of ChangePoints]. a numeric scalar; the mean number of seasonal changepoints.
• ncpPr: [Probability of the Number of ChangePoints]. A vector of length (scp.minmax[2]+1)=scp.max+1. It gives a probability distribution of having a certain number of seasonal changepoints over the range of [0,scp.max]; for example, ncpPr[1] is the probability of having no seasonal changepoint; ncpPr[i] is the probability of having (i-1) changepoints: Note that the index is i rather than (i-1) because ncpPr[1] is used for having zero changepoint.
• cpOccPr: [ChangePoint OCCurrence PRobability]. a vector of length N; it gives a probability distribution of having a changepoint in the seasonal component at each point of time. Plotting cpOccPr will depict a continuous
curve of probability-of-being-changepoint over the time. Of particular note, in the curve, a higher value at a peak indicates a higher chance of being a changepoint only at that particular SINGLE point in time, and does not necessarily mean a higher chance of observing a changepoint AROUND that time. For example, a window of cpOccPr values $c(0, 0, 0.5, 0, 0)$ (i.e., the peak prob is 0.5 and the summed prob is 0.5) is less likely to be a changepoint compared to another window values $c(0.1, 0.2, 0.3, 0.2, 0.1)$ (i.e., the peak prob is 0.3 but the summed prob is 0.8).

- **order**: a vector of length $N$; the average harmonic order needed to approximate the seasonal component. As an average over many sampled individual piece-wise harmonic curves, order is not necessarily an integer.
- **cp**: [Changepoints] a vector of length scp.max=scp.minmax[2]; the most possible changepoint locations in the seasonal component. The locations are obtained by first applying a sum-filtering to the cpOccPr curve with a filter window size of sseg.min and then picking up to a total ncp of the highest peaks in the filtered curve. If ncp<scp.max, the remaining of the vector is filled with NaNs.
- **cpPr**: [Changepoints PRobability] a vector of length scp.max; the probabilities associated with the changepoints cp. Filled with NaNs for the remaining elements if ncp<scp.max.
- **cpCI**: [Changepoints Credible Interval] a matrix of dimension scp.max x 2; the credible intervals for the detected changepoints cp.
- **cpAbruptChange**: [Abrupt change at Changepoints] a vector of length scp.max; the jumps in the fitted trend curves at the detected changepoints cp.
- **Y**: a vector of length $N$; the estimated trend component. It is the Bayesian model averaging of all the individual sampled trend.
- **SD**: [Standard Deviation] a vector of length $N$; the estimated standard deviation of the estimated trend component.
- **CI**: [Standard Deviation] a matrix of dimension $N$ x 2; the estimated credible interval of the estimated trend. One vector of the matrix is for the upper envelope and another for the lower envelope.
- **amp**: [AMPplitude] a vector of length $N$; the time-varying amplitude of the estimated seasonality.
- **ampSD**: [Standar Deviation of AMPplitude] a vector of length $N$; , the SD of the amplitude of the seasonality.
• pos_cpAbruptChange:
• neg_cpAbruptChange:
• pos_cpCI:
• neg_cpCI: The above variables have the same outputs as those variables without the prefix 'pos' and 'neg', except that we differentiate the change-points with a POSitive jump in the trend from those changepoints with a NEGative jump. For example, pos_ncp refers to the average number of trend changepoints that jump up (i.e., positively) in the trend.

References


See Also

beast, beast.irreg, beast123, minesweeper, tetris, geelandsat

Examples

library(Rbeast)

#################################Example 1############################################
# 'googletrend_beach' is the monthly Google search popularity of 'beach' in the US
# from 2004 to 2022, obtained from Google Trends. Sudden changes in the time series
# coincide with known extreme weather events (e.g., 2006 North American Blizzard, 2011
# US hottest summer on record, Record warm January in 2016) or the covid19 outbreak

o <- beast(googletrend_beach)

plot(o)
plot(o, vars=c('t','slpsgn') ) # plot the trend and probability of slope sign only.
# In the slpsgn panel, the upper red portion refers to
# probability of trend slope being positive, the middle
# green to the prob of slope being zero, and the lower
# blue to the probability of slope being negative.

#################################Example 1############################################
# Yellowstone is a half-monthly satellite time series of 774 NDVI (vegetation greeness) observations starting from July 1-15,1981 (i.e., start=c(1981,7,7)) at a Yellowstone site. It has 24 data points per year (i.e., freq=24). Note that the beast function handles only evenly-spaced regular time series. Irregular data need to be first aggregated at a regular time interval of your choice—the aggregation functionality is implemented in beast.irreg() and beast123().

data(Yellowstone)
plot(Yellowstone)

# Yellowstone is not a object of class 'ts' but a pure vector without time attributes. Below, no extra argument is supplied, so default values (i.e., start=1, deltat=1) are used and the time is 1:774. 'freq' is missing and so is guessed via auto-correlation. Use of auto-correlation to compute the period of a cyclic time series is not always reliable, so it is suggested to always supply 'freq' directly, as in Example 2 and Example 3.

out = beast(Yellowstone)  # the times assumed to be 1:length(Yellowstone) by default
plot(out)

#########################################################################Example 2#########################################################################
# The time info such as start, delta, and freq is explicitly provided. 'start' can be given as (1) a vector comprising year, month, & day, (2) a fractional number, or (3) a R's Date. The unit of start and deltat does not necessarily refer to time and can be arbitrary (e.g., a sequence of data observed at evenly-spaced distances along a transect or an elevation gradient)

o=beast(Yellowstone, start = c(1981,7,7), deltat=1/24, freq=24)
o#o=beast(Yellowstone, start = 1981.5137, deltat=1/24, freq=24)
o#o=beast(Yellowstone, start = as.Date('1981-7-7'), deltat=1/24, freq=24)

print(o)  # o is a R LIST object with many fields
plot(o)  # plot many variables
plot(o, vars=c('st','s','t'))  # plot the Y, seasonal, and trend components only
plot(o, vars=c('s','scp','samp','t','tcp','tslp'))# plot some selected variables in 'o'. Type '?plot.beast' to see more about vars
plot(o, vars=c('s','t'), col=c('red','blue'))  # specify colors of selected subplots

plot(o$time, o$season$Y,type='l')  # directly plot output: the fitted season
plot(o$time, o$season$scpOccPr)  # directly plot output: season chgpt prob
plot(o$time, o$trend$Y,type='l')  # directly plot output: the fitted trend
plot(o$time, o$trend$scpOccPr)  # directly plot output: trend chgpt occurrence prob
plot(o$time, o$season$order)  # directly plot output: avg harmonic order

plot(o, interactive=TRUE)  # manually choose which variables to plot

#########################################################################Example 3#########################################################################
# Specify other arguments explicitly. Default values are used for missing parameters.
# Note that beast(), beast.irreg(), and beast123() call the same internal C/C++ library,
# so in beast(), the input parameters will be converted to metadata, prior, mcmc, and
# extra parameters as explained for the beast123() function. Or type 'View(beast)' to
# check the parameter assignment in the code.

out = beast(Yellowstone, freq=24, season='dummy', mcmc.samples=5000, tseg.min=20)
plot(out)

out=beast(
    Yellowstone, # Yellowstone: a pure numeric vector wo time info
    start=1981.51, deltat=1/24, freq = 24,
    season = 'harmonic', # periodic compnt exisits,fitted as a harmonic curve
    scp.minmax = c(0,3), # min and max numbers of seasonal changpts allowed
    sorder.minmax = c(1,5), # min and max harmonic orders allowed
    sseg.min = 24, # the min length of segments btw neighboring chnpts
    tcp.minmax = c(0,10),# min and max numbers of changpts allowed in the trend
    torder.minmax = c(0,1), # min and maxx polynomial orders to fit trend
    tseg.min = 24, # the min length of segments btw neighboring trend chnpts
    deseasonalize = TRUE, # remove the global seasonality before fitting the beast model
    detrend = TRUE, # remove the global trend before fitting the beast model
    mcmc.seed = 0, # a seed for mcmc
    mcmc.burnin = 500, # number of initial iterations discarded
    mcmc.chains = 2, # number of chains
    mcmc.thin = 3, # include samples every 3 iterations
    mcmc.samples = 6000 # number of samples taken per chain
)
plot(out)
plot(out, interactive=TRUE)

#################################Example 4############################################
# Run an interactive GUI to visualize how BEAST is samplinig from the possible model
# spaces in terms of the numbers and timings of seasonal and trend changepoints.
# The GUI interface allows changing the option parameters interactively. This GUI is
# only available on Win x64 machines, not Mac or Linux.

## Not run:
beast(Yellowstone, freq=24, gui=TRUE)
## End(Not run)

# Apply beast to trend-only data. 'Nile' is the ANNUAL river flow of the river
# Nile at Aswan since 1871. It is a 'ts' object; its time attributes (start=1871,
# end=1970,frequency=1) are used to replace the user-supplied start,deltat, and freq,
# if any.

data(Nile)
plot(Nile)
attributes(Nile) # a ts object with time attributes (i.e., tsp=(start,end,freq)
o = beast(Nile)  # start=1871, delta=1, and freq=1 taken from Nile itself
plot(o)

o = beast(Nile,     # the same as above. The user-supplied values (i.e., 2011,
   start=2011,    # 24434) are ignored bcz Nile has its own time attributes.
   freq =24434,   # Its frequency tag is 1 (i.e., trend-only), so season='none'
   season='harmonic' # is used instead of the supplied 'harmonic'
)

#################################Example 5############################################
# NileVec is a pure data vector. The first run below is WRONG bcz NileVec was assumed
# to have a periodic component by default and beast gets a best estimate of freq=6 while
# the true value is freq=1. To fit a trend-only model, season='none' has to be explicitly
# specified, as in the 2nd & 3rd funs.

NileVec = as.vector(Nile) # NileVec is not a ts obj but a pure numeric data vector
o = beast(NileVec)  # ERROR: No time attributes available to interpret NileVec.
   # By default, beast assumes season='harmonic', start=1, and
   # deltat=1. 'freq' is missing and guessed to be 6 (WRONG).
plot(o)

o=beast(NileVec,season='none') # Use season='none' for trend-only analysis; the default
   # time is the indices '1:length(NileVec)'.

o=beast(NileVec,
   start = 1871,    # The true time attributes, if needed, has to be given
   deltat = 1,      # explicitly thru start and deltat (or freq if there is
   season = 'none') # a cyclic cmpnt).
plot(o)

#################################Example 6############################################
# beast can handle missing data. co2 is a monthly time series (i.e.,freq=12) starting
# from Jan 1959. We generate some missing values at random indices

## Not run:
data(co2)
attributes(co2) # A ts object with time attributes (i.e., tsp)
badIdx = sample( 1:length(co2), 50) # Get a set of random indices
co2[badIdx] = NA # Insert some data gaps
out=beast(co2) # co2 is a ts object and its 'tsp' time attributes are used to get the
   # true time info. No need to specify 'start','deltat', & freq explicitly.

out=beast(co2,
   start = c(1959,1,15),# The supplied time/freq values will be ignored bcz
   deltat = 1/12,      # co2 is a ts object; the correct freq = 12 will be
   freq = 365)         # used.
print(out)
Example 7

Apply beast to time series-like sequence data: the unit of sequences is not necessarily time.

```r
data(CNAchrom11) # DNA copy number alterations in Chromosome 11 for cell line GM05296
# The data is ordered by genomic position (not time), and the values
# are the log2-based intensity ratio of copy numbers between the sample
# and the reference. A value of zero means no gain or loss in copy number.
o = beast(CNAchrom11, season='none') # season is a misnomer here because the data has nothing to do with time. Regardless, we fit only a trend.
plot(o)
```
plot(out) # the result is wrong bc the guessed freq via auto-correlation by beast
# is 2 rather than 12, so we recommend always specifying 'freq' explicitly
# for those time series with a periodic component, as shown below.
out = beast(births,start=c(1946,1,15), deltat=1/12, freq=12 )
plot(out)

#################################Example 10############################################
#
# Daily confirmed COVID-19 new cases and deaths across the globe
#
## Not run:
data(covid19)
newcases = covid19$newcases

# This ts varies periodically every 7 days. 7 days can't be precisely represented
# in the unit of year bcz some years has 365 days and others has 366. So, here we
# use the date number as the time unit--the num of days lapsed since 1970-01-01.
datenum = as.numeric(covid19$date)
o = beast(newcases, start=min(datenum), deltat=1, freq=7)
o$time = as.Date(o$time, origin='1970-01-01') # Convert from integers to Date.
plot(o)

# Apply BEAST to the square root-transformed time series
o = beast(sqrt(newcases), start=min(datenum), deltat=1, freq=7)
o$time = as.Date(o$time, origin='1970-01-01') # Convert from integers to Date.
plot(o)

## End(Not run)

#################################Example11############################################
#
# The old API interface of beast is still made available but not recommended. It is
# kept mainly to ensure the working of the sample code on Page 475 in the text
# Ecological Metods by Drs. Southwood and Henderson.
#
## Not run:

# The interface as shown here will be deprecated and not recommended.
beast(Yellowstone, 24) #24 is the freq: number of datapoints per period

# Specify the model or MCMC parameters through opt as in Rbeast v0.2
opt=list()
opt$period=24 #Period of the cyclic component (i.e., freq in the new version)
opt$minSeasonOrder=2 #Min harmonic order allowed in fitting season component
opt$maxSeasonOrder=8 #Max harmonic order allowed in fitting season component
opt$minTrendOrder=0 #Min polynomial order allowed to fit trend (0 for constant)
opt$maxTrendOrder=1 #Max polynomial order allowed to fit trend (1 for linear term)
opt$minSepDist_Season=20 #Min separation time btw neighboring season changepoints
opt$minSepDist_Trend=20 #Min separation time btw neighboring trend changepoints
opt$maxKnotNum_Season=4 #Max number of season changepoints allowed
beast.irreg

Bayesian time series decomposition for changepoint, trend, and periodicity or seasonality

Description

A Bayesian model averaging algorithm called BEAST to decompose time series or 1D sequential data into individual components, such as abrupt changes, trends, and periodic/seasonal variations. BEAST is useful for changepoint detection (e.g., breakpoints or structural breaks), nonlinear trend analysis, time series decomposition, and time series segmentation.

Usage

beast.irreg(
  y,
  time, deltat,
  freq = NA,
  season = c('harmonic','dummy','none'),
  scp.minmax = c(0,10), sorder.minmax = c(0,5), sseg.min = NULL,
  tcp.minmax = c(0,10), torder.minmax = c(0,1), tseg.min = NULL,
  detrend = FALSE,
  deseasonalize= FALSE,
  mcmc.seed = 0,
  mcmc.burnin = 200,
  mcmc.chains = 3,
mcmc.thin = 5,
mcmc.samples = 8000,
ci = FALSE,
precValue = 1.5,
precPriorType = c('componentwise','uniform','constant','orderwise'),
print.options = TRUE,
print.progress = TRUE,
gui = FALSE,
...)

Arguments

y a vector for an irregular or unordered time series. Missing values such as NA and NaN are allowed.

- If y is regular (i.e., evenly-spaced in time), use the beast function instead.
- If y is a matrix or 3D array consisting of multiple regular or irregular time series (e.g., stacked images), use beast123 instead.

If a list of multiple time series is provided for y, the multivariate version of the BEAST algorithm will be invoked to decompose the multiple time series and detect common changepoints altogether. This feature is experimental only and under further development. Check ohio for a working example.

time a vector of the same length as y; the time vector of datapoints of y. It can be a numeric vector or a vector of Dates. If your time is formatted as strings or vectors of year, months, and days, use beast123() instead where more formats are supported.

deltat numeric; a user-specified time interval to aggregate y based on time into a regular time series. The BEAST model is currently formulated for regular data only, so internally, the beast.irreg function will aggregate/re-bin irregular data into regular ones; for the aggregation, the deltat MUST be provided to specify the desired bin size or time interval. The unit of delta needs to be consistent with time. If time takes a numeric vector, the unit is arbitrary and irrelevant to beast. If time takes a vector of Dates, the unit is assumed to YEAR; for example, if the desired time interval is 1 month (or 1 day), deltat should be 1/12 (or 1/365).

freq numeric. Needed only for data with a periodic/cyclic component (i.e., season='harmonic' or 'dummy') and ignored for trend-only data (i.e., season='none'). It specifies the number of samples/datapoints per cycle (e.g., a time series of monthly observations with an annual period has a frequency of 12); it may be a decimal real number (e.g., a time series of bi-hourly observations with a period of 37.5 hrs has a freq of 37.5/2=18.75). The period of the cyclic component in the unit of time is period=deltat*freq. freq is not a model parameter of BEAST and it has to be specified by the user. If freq is missing, BEAST first attempts to guess its value via auto-correlation before fitting the model, but in this case, freq is assumed to be an integer.

season characters (default to 'harmonic'); specify if y has a periodic component or not. Three strings are possible.
• 'none': y is trend-only; no periodic components are present in the time series. The args for the seasonal component (i.e., sorder.minmax, scp.minmax and sseg.max) will be ignored.

• 'harmonic': y has a periodic/seasonal component. The term 'season' is a misnomer, being used here to broadly refer to any periodic variations present in y. The periodicity is NOT a model parameter estimated by beast but a known constant given by the user through freq. By default, the periodic component is modeled as a harmonic curve—a combination of sines and cosines.

• If 'dummy', the same as 'harmonic' except that the periodic/seasonal component is modeled as a non-parametric curve. The arg sorder.minmax is irrelevant and is ignored.

sorder.minmax a vector of 2 integers (>=1); the min and max harmonic orders considered to fit the seasonal component. sorder.minmax is used only if the time series has a seasonal component (i.e., season='harmonic') and ignored for trend-only data or when season='dummy'. If the min and max orders are equal (sorder.minmax[1]=sorder.minmax[2]), BEAST assumes a constant harmonic order used and won’t infer the posterior probability of harmonic orders.

scp.minmax a vector of 2 integers (>=0); the min and max number of seasonal changepoints (scp) allowed in segmenting the seasonal component. scp.minmax is used only if y has a seasonal component (i.e., season='harmonic' or 'dummy') and ignored for trend-only data. If the min and max changepoint numbers are equal, BEAST assumes a constant number of scp and won’t infer the posterior probability of the number of changepoints, but it still estimates the occurrence probability of the changepoints over time (i.e., the most likely times at which these changepoints occur). If both the min and max numbers are set to 0, no changepoints are allowed; then a global harmonic model is used to fit the seasonal component, but still, the most likely harmonic order will be inferred if sorder.minmax[1] is not equal to sorder.minmax[2].

sseg.min an integer; the min segment length allowed between two neighboring season changepoints. That is, when fitting a piecewise harmonic seasonal model, no two changepoints are allowed to occur within a time window of length sseg.min. sseg.min must be a unitless integer—the number of time intervals/data points so that the time window in the original unit is sseg.min*deltat. sseg.min defaults to NULL and its value will be given a default value in reference to freq.

torder.minmax a vector of 2 integers (>=0); the min and max orders of the polynomials considered to fit the trend component. The 0-th order corresponds to a constant term/a flat line and the 1st order is a line. If torder.minmax[1]=torder.minmax[2], BEAST assumes a constant polynomial order used and won’t infer the posterior probability of polynomial orders.

tcp.minmax a vector of 2 integers; the min and max number of trend changepoints (tcp) allowed in segmenting the trend component. If the min and max changepoint numbers are equal, BEAST assumes a constant number of changepoints and won’t infer the posterior probability of the number of changepoints for the trend, but it still estimates the occurrence probability of the changepoints over time (i.e., the most likely times at which these changepoints occur in the trend). If both the min
and max numbers are set to 0, no changepoints are allowed; then a global poly-
nomial trend is used to fit the trend component, but still, the most likely poly-
nomial order will be inferred if torder.minmax[1] is not equal to torder.minmax[2].

**tseg.min**
an integer; the min segment length allowed between two neighboring trend
changepoints. That is, when fitting a piecewise polynomial trend model, no
two changepoints are allowed to occur within a time window of length tseg.min.
tseg.min must be an unitless integer—the number of time intervals/data points
so that the time window in the original unit is tseg.min*deltat. tseg.min
defaults to NULL and its value will be given a default value in reference to freq
if the time series has a cyclic component.

**detrend**
logical; If TRUE, a global trend is first fitted and removed from the time series
before running BEAST; after BEAST finishes, the global trend is added back to
the BEAST result.

**deseasonalize**
logical; If TRUE, a global seasonal model is first fitted and removed from the time
series before running BEAST; after BEAST finishes, the global seasonal curve is
added back to the BEAST result. deseasonalize is ignored if season='none'
(i.e., trend-only data).

**mcmc.seed**
integer (>=0); the seed for the random number generator used for Monte Carlo
Markov Chain (mcmc). If mcmc.seed=0, an arbitrary seed is picked and the
fitting results vary across runs. If fixed to the same non-zero integer, the result
can be re-produced for different runs. But the results from the same seed may
still vary if run on different computers because the random generator library
depends on CPU's instruction sets.

**mcmc.chains**
in integer (>0); the number of MCMC chains.

**mcmc.thin**
integer (>0); a factor to thin chains (e.g., if thinningFactor=5, samples will be
taken every 3 iterations)

**mcmc.burnin**
in integer (>0); the number of burn-in samples discarded at the start of each chain

**mcmc.samples**
in integer (>=0); the number of samples collected per MCMC chain. The total
number of iterations is (burnin+samples*thin)*chains.

**ci**
boolean; If TRUE, credible intervals (i.e., out$season$CI or out$trend$CI) will
be computed for the estimated seasonal and trend components. Computing CI
is time-consuming, due to sorting, so set ci to FALSE if a symmetric credible
interval (i.e., out$trend$SD and out$season$SD) suffices.

**precValue**
numeric (>0); the hyperparameter of the precision prior; the default value is
1.5. precValue is useful only when precPriorType='constant', as further ex-
plained below

**precPriorType**
characters. It takes one of 'constant', 'uniform', 'componentwise' (the default),
and 'orderwise'. Below are the differences between them.

1. 'constant': the precision parameter used to parameterize the model co-
efficients is fixed to a constant specified by precValue. In other words,
precValue is a user-defined hyperparameter and the fitting result may be
sensitive to the chosen values of precValue.

2. 'uniform': the precision parameter used to parameterize the model coeffi-
cients is a random variable; its initial value is specified by precValue. In
other words, precValue will be inferred by the MCMC, so the fitting result
will be insensitive to the choice in precValue.
3. 'componentwise': multiple precision parameters are used to parameterize the model coefficients for individual components (e.g., one for season and another for trend); their initial values is specified by precValue. In other words, precValue will be inferred by the MCMC, so the fitting result will be insensitive to the choice in precValue.

4. 'orderwise': multiple precision parameters are used to parameterize the model coefficients not just for individual components but also for individual orders of each component; their initial values is specified by precValue. In other words, precValue will be inferred by the MCMC, so the fitting result will be insensitive to the choice in precValue.

print.options boolean. If TRUE, the full list of input parameters to BEAST will be printed out prior to the MCMC inference; the naming for this list (e.g., metadata, prior, and mcmc) differs slightly from the input to beast, but there is a one-to-one correspondence (e.g., prior$strendMinSepDist=tseg.min). Internally, beast converts the input parameters to the forms of metadata, prior, and mcmc. Type 'View(beast)' to see the details or check the beast123 function.

print.progress boolean; If TRUE, a progressbar will be displayed.

gui boolean. If TRUE, BEAST will be run in a GUI demonstration mode, with a GUI window to show an animation of the MCMC sampling in the model space step by step. Note that "gui=TRUE" works only for Windows x64 systems not Windows 32 or Linux/Mac systems.

... additional parameters. There are many more settings for the implementation but not made available in the beast() interface; please use the function beast123() instead

Value

The output is an object of class "beast". It is a list, consisting of the following variables. In the explanations below, we assume the input y is a single time series of length N:

time a vector of size 1xN: the times at the N sampled locations. By default, it is simply set to 1:N
data a vector, matrix, or 3D array; this is a copy of the input data if extra$dumpInputData = TRUE. If extra$dumpInputData=FALSE, it is set to NULL. If the original input data is irregular, the copy here is the regular version aggregated from the original at the time interval specified by metadata$deltaTime.
marg_lik numeric; the average of the model marginal likelihood; the larger marg_lik, the better the fitting for a given time series.
R2 numeric; the R-square of the model fitting.
RMSE numeric; the RMSE of the model fitting.
sig2 numeric; the estimated variance of the model error.
trend a list object consisting of various outputs related to the estimated trend component:
  • ncp: [Number of ChangePoints]. a numeric scalar; the mean number of trend changepoints. Individual models sampled by BEAST has a varying
dimension (e.g., number of changepoints or knots), so several alternative statistics (e.g., ncp_mode, ncp_median, and ncp_pct90) are also given to summarize the number of changepoints. For example, if nmc$c.samples=10, the numbers of changepoints for the 10 sampled models are assumed to be c(0, 2, 4, 1, 1, 2, 7, 6, 6, 1). The mean ncp is 3.1 (rounded to 3), the median is 2.5 (2), the mode is 1, and the 90th percentile (ncp_pct90) is 6.5.

• ncp_mode: [Number of ChangePoints]. a numeric scalar; the mode for number of changepoints. See the above for explanations.

• ncp_median: [Number of ChangePoints]. a numeric scalar; the median for number of changepoints. See the above for explanations.

• ncp_pct90: [Number of ChangePoints]. a numeric scalar; the 90th percentile for number of changepoints. See the above for explanations.

• ncpPr: [Probability of the Number of ChangePoints]. A vector of length (tcp.minmax[2]+1)=tcp.max+1. It gives a probability distribution of having a certain number of trend changepoints over the range of [0,tcp.max]; for example, ncpPr[1] is the probability of having no trend changepoint; ncpPr[i] is the probability of having (i-1) changepoints: Note that it is ncpPr[i] not ncpPr[i-1] because ncpPr[1] is used for having zero change-point.

• cpOccPr: [ChangePoint OCCurrence PRObability]. a vector of length N; it gives a probability distribution of having a changepoint in the trend at each point of time. Plotting cpOccPr will depict a continuous curve of probability-of-being-change-point. Of particular note, in the curve, a higher peak indicates a higher chance of being a changepoint only at that particular SINGLE point in time and does not necessarily mean a higher chance of observing a changepoint AROUND that time. For example, a window of cpOccPr values c(0, 0, 0.5, 0, 0) (i.e., the peak prob is 0.5 and the summed prob is 0.5) is less likely to be a changepoint compared to another window c(0.1, 0.2, 0.21, 0.2, 0.1) (i.e., the peak prob is 0.21 but the summed prob is 0.71).

• order: a vector of length N; the average polynomial order needed to approximate the fitted trend. As an average over many sampled individual piece-wise polynomial trends, order is not necessarily an integer.

• cp: [Changepoints] a vector of length tcp.max=tcp.minmax[2]; the most possible changepoint locations in the trend component. The locations are obtained by first applying a sum-filtering to the cpOccPr curve with a filter window size of tseg.min and then picking up to a total prior$MaxKnotNum/tcp.max of the highest peaks in the filtered curve. NaNs are possible if no enough changepoints are identified. cp records all the possible changepoints identified and many of them are bound to be false positives. Do not blindly treat all of them as actual changepoints.

• cpPr: [Changepoints PRObability] a vector of length tcp.max=tcp.minmax[2]; the probabilities associated with the changepoints cp. Filled with NaNs for the remaining elements if ncp<tcp.max.

• cpCI: [Changepoints Credible Interval] a matrix of dimension tcp.max x 2; the credible intervals for the detected changepoints cp.

• cpAbruptChange: [Abrupt change at Changepoints] a vector of length tcp.max; the jumps in the fitted trend curves at the detected changepoints cp.
\( \mathbf{Y} \): a vector of length N; the estimated trend component. It is the Bayesian model averaging of all the individual sampled trend.

- **SD**: [Standard Deviation] a vector of length N; the estimated standard deviation of the estimated trend component.

- **CI**: [Standard Deviation] a matrix of dimension \( N \times 2 \); the estimated credible interval of the estimated trend. One vector of the matrix is for the upper envelope and another for the lower envelope.

- **slp**: [Slope] a vector of length N; the time-varying slope of the fitted trend component.

- **slpSD**: [Standard Deviation of Slope] a vector of length N; the SD of the slope for the trend component.

- **slpSgnPosPr**: [Probability of slope having a positive sign] a vector of length N; the probability of the slope being positive (i.e., increasing trend) for the trend component. For example, if \( \text{slpSgnPosPr} = 0.80 \) at a given point in time, it means that 80% of the individual trend models sampled in the MCMC chain has a positive slope at that point.

- **slpSgnZeroPr**: [Probability of slope being zero] a vector of length N; the probability of the slope being zero (i.e., a flat constant line) for the trend component. For example, if \( \text{slpSgnZeroPr} = 0.10 \) at a given point in time, it means that 10% of the individual trend models sampled in the MCMC chain has a zero slope at that point. The probability of slope being negative can be obtained from \( 1 - \text{slpSgnPosPr} - \text{slpSgnZeroPr} \).

- **pos_ncp**:
  - **neg_ncp**:
  - **pos_ncpPr**:
  - **neg_ncpPr**:
  - **pos_cpOccPr**:
  - **neg_cpOccPr**:
  - **pos_cp**:
  - **neg_cp**:
  - **pos_cpPr**:
  - **neg_cpPr**:
  - **pos_cpAbruptChange**:
  - **neg_cpAbruptChange**:
  - **pos_cpCI**:
  - **neg_cpCI**: The above variables have the same outputs as those variables without the prefix 'pos' and 'neg', except that we differentiate the changepoints with a POSitive jump in the trend from those changepoints with a NEGative jump. For example, **pos_ncp** refers to the average number of trend changepoints that jump up (i.e., positively) in the trend.

- **inc_ncp**:
  - **dec_ncp**:
  - **inc_ncpPr**:
  - **dec_ncpPr**:
  - **inc_cpOccPr**:
  - **dec_cpOccPr**:
• dec_cp0ccPr:
• inc_cp:
• dec_cp:
• inc_cpPr:
• dec_cpPr:
• inc_cpAbruptChange:
• dec_cpAbruptChange:
• inc_cpCI:
• dec_cpCI: The above variables have the same outputs as those variables without the prefix 'inc' and 'dec', except that we differentiate the changepoints at which the trend slope increases from those changepoints at which the trend slope decreases. For example, if the trend slopes before and after a changpt is 0.4 and 2.5, then the changepoint is counted toward inc_ncp.

season a list object consisting of various outputs related to the estimated seasonal/periodic component:

• ncp: [Number of ChangePoints]. a numeric scalar; the mean number of seasonal changepoints.
• ncpPr: [Probability of the Number of ChangePoints]. A vector of length (scp.minmax[2]+1)=scp.max+1. It gives a probability distribution of having a certain number of seasonal changepoints over the range of [0,scp.max]; for example, ncpPr[1] is the probability of having no seasonal changepoint; ncpPr[1] is the probability of having (i-1) changepoints: Note that the index is i rather than (i-1) because ncpPr[1] is used for having zero changepoint.
• cp0ccPr: [ChangePoint OCCurence PRobability]. a vector of length N; it gives a probability distribution of having a changepoint in the seasonal component at each point of time. Plotting cp0ccPr will depict a continuous curve of probability-of-being-changepoint over the time. Of particular note, in the curve, a higher value at a peak indicates a higher chance of being a changepoint only at that particular SINGLE point in time, and does not necessarily mean a higher chance of observing a changepoint AROUND that time. For example, a window of cp0ccPr values c(0,0,0.5,0,0) (i.e., the peak prob is 0.5 and the summed prob is 0.5) is less likely to be a changepoint compared to another window values c(0.1,0.2,0.3,0.2,0.1) (i.e., the peak prob is 0.3 but the summed prob is 0.8).
• order: a vector of length N; the average harmonic order needed to approximate the seasonal component. As an average over many sampled individual piece-wise harmonic curves, order is not necessarily an integer.
• cp: [Changepoints] a vector of length scp.max=scp.minmax[2]; the most possible changepoint locations in the seasonal component. The locations are obtained by first applying a sum-filtering to the cp0ccPr curve with a filter window size of sseg.min and then picking up to a total ncp of the highest peaks in the filtered curve. If ncp<scp.max, the remaining of the vector is filled with NaNs.
• cpPr: [Changepoints PRObability] a vector of length scp.max; the probabilities associated with the changepoints cp. Filled with NaNs for the remaining elements if ncp<scp.max.
• \( cp\text{CI} \): [Change points Credible Interval] a matrix of dimension \( scp.\text{max} \times 2 \); the credible intervals for the detected changepoints \( cp \).

• \( cp\text{AbruptChange} \): [Abrupt change at Changepoints] a vector of length \( scp.\text{max} \); the jumps in the fitted trend curves at the detected changepoints \( cp \).

• \( Y \): a vector of length \( N \); the estimated trend component. It is the Bayesian model averaging of all the individual sampled trend.

• \( SD \): [Standard Deviation] a vector of length \( N \); the estimated standard deviation of the estimated trend component.

• \( CI \): [Standard Deviation] a matrix of dimension \( N \times 2 \); the estimated credible interval of the estimated trend. One vector of the matrix is for the upper envelope and another for the lower envelope.

• \( amp \): [AMPlitude] a vector of length \( N \); the time-varying amplitude of the estimated seasonality.

• \( ampSD \): [Standar Deviation of AMPlitude] a vector of length \( N \); the SD of the amplitude of the seasonality.

• \( pos\text{ncp} \):
• \( neg\text{ncp} \):
• \( pos\text{ncpPr} \):
• \( neg\text{ncpPr} \):
• \( pos\text{cpOccPr} \):
• \( neg\text{cpOccPr} \):
• \( pos\text{cp} \):
• \( neg\text{cp} \):
• \( pos\text{cpPr} \):
• \( neg\text{cpPr} \):
• \( pos\text{cpAbruptChange} \):
• \( neg\text{cpAbruptChange} \):
• \( pos\text{cpCI} \):

• \( neg\text{cpCI} \): The above variables have the same outputs as those variables without the prefix ‘pos’ and ‘neg’, except that we differentiate the changepoints with a POStive jump in the trend from those changepoints with a NEGative jump. For example, \( pos\text{ncp} \) refers to the average number of trend changepoints that jump up (i.e., positively) in the trend.

References


See Also

beast, beast123, minesweeper, tetris, geeLandsat

Examples

library(Rbeast)

#########################################################################
# Note that the BEAST algorithm is currently implemented to handle only regular time
# series. 'beast.irreg' accepts irregular time series but internally it aggregates them
# into regular ones prior to applying the BEAST model. For the aggregation, both the
# "time" and "deltat" args are needed to specify individual times of data points and the
# regular time interval desired. If there is a cyclic component, freq in the unit of
# time intervals/datapoints should also be given; if not, a possible value is guessed
# via auto-correlation

#########################################################################
# 'ohio' is a data.frame on an irregular Landsat time series of reflectances & ndvi
# (e.g., surface greenness) at an Ohio site. It has multiple columns of alternative date
# formats, such as year, month, day, doy (date of year), rdate (R's date class), and
# time (fractional year)

data(ohio)
str(ohio)

#########################################################################
# Below, 'time' is given as numeric values, which can be of any arbitrary unit. Although
# here 1/12 can be interpreted as 1/12 year or 1 month, BEAST itself doesn't care about
# the time unit. So, the unit of 1/12 is irrelevant for BEAST. 'freq' is missing and a
# guess of it is used.

o=beast.irreg(ohio$ndvi, time=ohio$time, deltat=1/12)
plot(o)
print(o)

#########################################################################
# Aggregate the time series at a monthly interval (deltat=1/12) and explicitly provide
# the 'freq' arg

o=beast.irreg(ohio$ndvi, time=ohio$time, deltat=1/12, freq=12)

#########################################################################
# Aggregate the time series at a half-monthly time interval, and the freq becomes 24,
# that is, PERIOD (1 year) = deltat (1/24 year) X freq (24)

do=beast.irreg(ohio$ndvi, time=ohio$time,deltat=1/24, freq=24)

# 'time' is given as R's dates. The unit is YEAR. 1/12 refers to 1/12 year or 1 month

do=beast.irreg(ohio$ndvi, time=ohio$rdate,deltat=1/12)

# If the input time has formats (e.g., date strings) other than the above two, use the
# beast123() function instead.

beast123

Bayesian time series decomposition for changepoint, trend, and periodicity or seasonality

Description

A Bayesian model averaging algorithm called BEAST to decompose time series or 1D sequential
data into individual components, such as abrupt changes, trends, and periodic/seasonal variations.
BEAST is useful for changepoint detection (e.g., breakpoints or structural breaks), nonlinear trend
analysis, time series decomposition, and time series segmentation.

Usage

beast123( Y,
    metadata=list(),
    prior =list(),
    mcmc =list(),
    extra =list(),
    season =c('harmonic','dummy','none'),
    ...
)

Arguments

Y

a 1D vector, 2D matrix, or 3D array of numeric data. Missing values are allowed
and can be indicated by NA, NaN, or a value customized in the 2nd argument
metadata (e.g., metadata$missingValue=-9999).

- If Y is a vector of size N x 1 or 1 x N, it is treated as a single time series of
  length N.
- If Y is a 2D matrix or 3D array of dimension N1 x N2 or N1 x N2 x N3 (e.g.,
  stacked images of geospatial data), it includes multiple time series of equal
length: Which dimension is time has to be specified in the 2nd argument using \texttt{metadata$whichDimIsTime}. For example, \texttt{metadata$whichDimIsTime} = 1 for a 190x35 2D input indicates 35 time series of length 190 each; \texttt{metadata$whichDimIsTime} = 2 for a 100x200x300 3D input indicates 30000=100*300 time series of length 200 each.

\texttt{Y} can be either regular (i.e., evenly-spaced in time) or irregular/unordered in time.

- If regular, individual times are determined from the time of the 1st data point \texttt{startTime} and the time span between consecutive points \texttt{deltaTime}, which are specified in the 2nd arg through \texttt{metadata$startTime} and \texttt{metadata$deltaTime}; if not given, \texttt{startTime} and \texttt{deltaTime} take a default 1.0.
- If irregular or regular but unordered, the times have to be explicitly given through \texttt{metadata$time}. The BEAST model is currently formulated for regular data only, so internally, the \texttt{beast123} function will aggregate/re-bin irregular data into regular ones; for the aggregation, the \texttt{metadata$deltaTime} parameter MUST also be also provided to specify the desired bin size or time interval.

\texttt{Y} can have a periodic component or have a trend component only. Use the argument \texttt{season} to specify the cases.

- If \texttt{season='none'}, \texttt{Y} is treated as trend-only; no periodic components are present in the time series.
- If \texttt{season='harmonic'}, \texttt{Y} has a periodic/seasonal component. The term 'season' is a misnomer being used here to broad refer to any periodic variations present in \texttt{Y}. The periodicity is not a statistical parameter estimated by BEAST but a known constant given by the user through \texttt{metadata$freq}. The periodic component is modeled as a harmonic curve—a combination of sines and cosines.
- If \texttt{season='dummy'}, the same as 'harmonic' except that the periodic/seasonal component is modeled as a non-parametric curve.

\textbf{metadata} (optional). If present, \texttt{metadata} can be either an INTEGER specifying the known period of the cyclic/seasonal component or a LIST specifying various parameters to describe the 1st argument \texttt{Y}. If missing, default values will be used, but \texttt{metadata} must be explicitly specified if the input \texttt{Y} is a 2D matrix or 3D array. \texttt{metadata} is not part of BEAST's Bayesian formulation but just some additional info to interpret \texttt{Y}. Below are possible fields in \texttt{metadata}; not all of them are always needed, depending on the types of inputs (e.g., 1D, 2D or 3D; regular or irregular).

- \texttt{metadata$whichDimIsTime}: integer (\leq 3). Needed to specify which dimension of \texttt{Y} is time for a matrix or 3D array input. Ignored if the input \texttt{Y} is a vector.
- \texttt{metadata$isRegularOrdered}: logical (default to \texttt{TRUE} if missing). If \texttt{TRUE}, the 1st argument \texttt{Y} is assumed to be regular and if \texttt{FALSE}, \texttt{Y} is irregular or regular but unordered in time.
- \texttt{metadata$time}: numeric or string values to specify the times for irregular \texttt{Y}. Needed ONLY if \texttt{isRegularOrdered} = \texttt{FALSE} (i.e. irregular inputs). Ignored if \texttt{isRegularOrdered} = \texttt{TRUE} (i.e., regular data for which
metadata$startTime and metadata$deltaTime are used instead). Three ways of specifying metadata$time are supported:

1. a vector of numerical values to indicate times. The unit of the times is irrelevant to BEAST as long as it remains consistent as the unit used for specifying other variables such as startTime and deltaTime. The length of the metadata$time vector must be the same as Y’s time dimension.

2. a list of vectors to specify individual dates of the time series. Use metadata$time$year, metadata$time$month, and metadata$time$day to give the dates; or alternatively use metadata$time$year and metadata$time$doy where each value of the doy vector is a number within 1 and 365/366. Each vector must have the same length as the time dimension of Y.

3. a vector of date strings. Use metadata$time$dateStr for the date strings, and metadata$time$strFmt to specify the format for parsing dateStr. Three formats are currently supported:

   – (a). All the date strings have a fixed pattern in terms of the relative positions of year, month, and day. For example, to extract 2001/12/02 etc from metadata$time$dateStr = c('P23R34-2001.1202333xd', 'O93X94-2002.1108133fd', 'TP3R34-2009.0122333td') use strFmt='P23R34-yyyy.mmdd333xd' where yyyy, mm, and dd are the specifiers and other positions are wildcards and can be filled with any other letters different from yyyy, mm and dd.

   – (b). All the date strings have a fixed pattern in terms of the relative positions of year and doy. For example, to extract 2001/045(day of year) from 'P23R342001888045', use strFmt='123123yyyy888doy' where yyyy and doy are the specifiers and other positions are wildcards and can be filled with any other letters different from yyyy, and doy. 'doy' must be three digit in length.

   – (c). All the date strings have a fixed pattern in terms of the separation characters between year, month, and day. For example, to extract 2002/12/02 from '2002,12/02', ' 2002 , 12/2', '2002,12 /02 ', use strFmt='Y,M/D' where the whitespaces are ignored. To get 2002/12/02 from '2–12, 2012 ', use strmFmt='D–M,Y'.

• metadata$startTime: numeric (default to 1.0 if missing). It gives the time of the 1st data point. It can be specified as a scalar (e.g., 2021.23) or a vector of three values in the order of year, month, and day (e.g., metadata$startTime = c(2021,1,24)). metadata$startTime is needed for regular input data but optional for irregular data: If missing, startTime will be computed from metadata$time for irregular Y.

• metadata$deltaTime: numeric. It specifies the time interval between consecutive data points. It is optional for regular data (default to 1.0 if not supplied), but has to be specified for irregular data because deltaTime is needed to aggregate/resample the irregular time series into regular ones.

• metadata$freq: integer. Needed only for data with a periodic/cyclic component (i.e., season=’harmonic’ or ’dummy’ ) and ignored for trend-only data (i.e., season=’none’ ). The ”freq” parameter must be an INTEGER specifying the number of samples/values/points per cycle (e.g, a monthly time series with an annual period has a frequency of 12. If freq is absent,
BEAST first attempts to guess its value via auto-correlation before fitting the model.

- `metadata$missingValue`: numeric; a customized value to indicate bad/missing values in the time series, in addition to those NA or NaN values.
- `metadata$maxMissingRate`: a fractional number within [0, 1] as the maximum percentage of missing values, above which the time series will be skipped and won’t be fitted by BEAST.

`prior` (optional). a list object consisting of the hyperprior parameters in the Bayesian formulation of the BEAST model. Because they are part of the model, the fitting result may be sensitive to the choices of these hyperparameters. If `prior` is missing, a set of default values will be used and the exact values used will be printed to the console at the start of the BEAST run. Below are possible parameters:

- `prior$seasonMinOrder`: integer (>=1)
- `prior$seasonMaxOrder`: integer (>=1); the min and max harmonic orders considered to fit the seasonal component. `seasonMinOrder` and `seasonMaxOrder` are only used if the time series has a seasonal component (i.e., `season='harmonic'`) and ignored for trend-only data or when `season='dummy'`. If `seasonMinOrder=seasonMaxOrder`, BEAST assumes a constant harmonic order used and won’t infer the posterior probability of harmonic orders.
- `prior$seasonMinKnotNum`: integer (>=0)
- `prior$seasonMaxKnotNum`: integer (>=0); the min and max number of seasonal changepoints allowed in segmenting and fitting the seasonal component. `seasonMinKnotNum` and `seasonMaxKnotNum` are only used if the time series has a seasonal component (i.e., `season='harmonic'` or `season='dummy'`) and ignored for trend-only data. If `seasonMinOrder=seasonMaxOrder`, BEAST assumes a constant number of changepoints and won’t infer the posterior probability of the number of changepoints, but it will still estimate the occurrence probability of the changepoints over time (i.e., the most likely times at which these changepoints occur). If `seasonMinOrder=seasonMaxOrder=0`, no changepoints are allowed in the seasonal component; then a global harmonic model is used to fit the seasonal component.
- `prior$seasonMinSepDist`: integer (>0), the min separation time between two neighboring season changepoints. That is, when fitting a piecewise harmonic seasonal model, no two changepoints are allowed to occur within a time window of `seasonMinSepDist`. `seasonMinSepDist` must be an unitless integer—the number of time intervals/data points so that the time window in the original unit is `seasonMinSepDist*metadata$deltaTime`.
- `prior$trendMinOrder`: integer (>=0)
- `prior$trendMaxOrder`: integer (>=0); the min and max orders of the polynomials considered to fit the trend component. The zero-th order corresponds to a constant term/ a flat line and the 1st order is a line. If `trendMinOrder=trendMaxOrder`, BEAST assumes a constant polynomial order used and won’t infer the posterior probability of polynomial orders.
- `prior$trendMinKnotNum`: integer (>=0)
- `prior$trendMaxKnotNum`: integer (>=0); the min and max number of trend changepoints allowed in segmenting and fitting the trend component. If
trendMinOrder=trendMaxOrder. BEAST assumes a constant number of changepoints in the fitted trend and won’t infer the posterior probability of the number of trend changepoints, but it will still estimate the occurrence probability of the changepoints over time (i.e., the most likely times at which these changepoints occur). If trendMinOrder=trendMaxOrder=0, no changepoints are allowed in the trend component; then a global polynomial model is used to fit the trend.

• prior\$trendMinSepDist: integer (>0). the min separation time between two neighboring trend changepoints.
• prior\$precValue: numeric (>0); the default value is 10.
• prior\$precPriorType: characters. It takes one of 'constant', 'uniform' (the default), 'componentwise', and 'orderwise'. Below are the differences between them.

1. precPriorType='constant': the precision parameter used to parameterize the model coefficients is fixed to a constant specified by prior\$precValue. In other words, prior\$precValue is a user-defined hyperparameter and the fitting result may be sensitive to the chosen values of prior\$precValue.
2. precPriorType='uniform': the precision parameter used to parameterize the model coefficients is a random variable; its initial value is specified by prior\$precValue. In other words, precValue will be inferred by the MCMC, so the fitting result is insensitive to the choice in prior\$precValue.
3. precPriorType='componentwise': multiple precision parameters are used to parameterize the model coefficients for individual components (e.g., one for season and another for trend); their initial values is specified by prior\$precValue. In other words, precValue will be inferred by the MCMC, so the fitting result is insensitive to the choice in prior\$precValue.
4. precPriorType='orderwise': multiple precision parameters are used to parameterize the model coefficients not just for individual components but also for individual orders of each component; their initial values is specified by prior\$precValue. In other words, precValue will be inferred by the MCMC, so the fitting result is insensitive to the choice in prior\$precValue.

\textbf{mcmc} (optional). a list object consisting of parameters to configure the MCMC inference. These parameter are not part of the Bayesian formulation of the BEAST model but are the settings for the reversible-jump MCMC to generate MCMC chains. Due to the MCMC nature, the longer the simulation chain is, the better the fitting result. Below are possible parameters:

• mcmc\$seed: integer (>=0); the seed for the random number generator. If mcmc\$seed=0, an arbitrary seed will be picked up and the fitting result will vary across runs. If fixed to the same on-zero integer, the results can be re-produced for different runs. Note that the results may still vary if run on different computers with the same seed because the random generator library depends on CPU’s instruction sets.
• mcmc\$samples: integer (>0); the number of samples collected per MCMC chain.
• `mcmc$chainNumber`: integer (>0); the number of parallel MCMC chains.
• `mcmc$thinningFactor`: integer (>0); a factor to thin chains (e.g., if thinningFactor=5, samples will be taken every 3 iterations).
• `mcmc$burnin`: integer (>0); the number of burn-in samples discarded at the start of each chain.
• `mcmc$maxMoveStepSize`: integer (>0). The RJMCMC sampler employs a move proposal when traversing the model space or proposing new positions of change points. ‘maxMoveStepSize’ is used in the move proposal to specify the max window allowed in jumping from the current changepoint.
• `mcmc$seasonResamplingOrderProb`: a fractional number less than 1.0; the probability of selecting a re-sampling proposal (e.g., resample seasonal harmonic order).
• `mcmc$trendResamplingOrderProb`: a fractional number less than 1.0; the probability of selecting a re-sampling proposal (e.g., resample trend polynomial order)
• `mcmc$credIntervalAlphaLevel`: a fractional number less than 1.0 (default to 0.95); the level of confidence used to compute credible intervals.

extra (optional). a list object consisting of flags to control the outputs from the BEAST runs or configure other program setting. Below are possible parameters:

• `extra$dumpInputData`: logical (default to FALSE). If TRUE, the input time series will be copied into the output. When the input Y is irregular (i.e., metadata$regularOrdered=FALSE), the dumped copies will be the aggregated regular time series.
• `extra$whichOutputDimIsTime`: integer (<=3). If the input Y is a 2D or 3D array (i.e., multiple time series such as stacked images), the whichOutputDimIsTime specifies which dimension is the time in the output variables. whichOutputDimIsTime defaults to 3 for 3D inputs and is ignored if the input is a vector (i.e., a single time series).
• `extra$ncpStatMethod`: character (deprecated). A string to specify which statistic is used to determine the Number of ChangePoint (ncp) when computing the most likely changepoint locations (e.g., out$trend$cp, and out$season$cp). Three values are possible: ‘mode’, ‘mean’, and ‘median’; the default is ‘mode’. Individual models sampled by BEAST has a varying dimension (e.g., number of change points or knots). For example, if mcmc$samples=10, the numbers of changepoints for the 10 sampled models are assumed to be c(0, 2, 4, 1, 1, 2, 7, 6, 6, 1). The mean ncp is 3.1 (rounded to 3), the median is 2.5 (2), and the mode is 1. This argument is deprecated; now all the possible changepoints are outputted, together with several versions of ncp, including ncp, ncp_median, ncp_mode, and ncp_pct90. A similar parameter ncpStat is added to the plot.beast function to specify which ncp is used when plotting.
• `extra$computeCredible`: logical (default to TRUE). Credible intervals will be computed and outputted only if set to TRUE.
• `extra$fastCIComputation`: logical (default to TRUE). If TRUE, a fast method is used to compute credible intervals (CI). Computation of CI is one of the most computational parts and fastCIComputation should be set to TRUE unless more accurate CI estimation is desired.
• `extra$computeSeasonOrder`: logical (default to TRUE). If TRUE, a posterior estimate of the seasonal harmonic order will be outputted; this flag is only valid if the time series has a seasonal component (i.e., `season='harmonic'` and `prior$seasonMinOrder` is not equal to `prior$seasonMaxOrder`).

• `extra$computeTrendOrder`: logical (default to TRUE). If TRUE, a posterior estimate of the trend polynomial order will be outputted; this flag is only valid when `prior$trendMinOrder` is not equal to `prior$trendMaxOrder`.

• `extra$computeSeasonChngpt`: logical (default to TRUE). If TRUE, compute the most likely times/positions where changepoints occur in the seasonal component. This flag is not valid if there is a seasonal component in the time series (i.e., `season='harmonic'` or `season='dummy'` and `prior$seasonMaxKnotNum` is non-zero).

• `extra$computeTrendChngpt`: logical (default to TRUE). If TRUE, compute the most likely times/positions where changepoints occur in the trend component.

• `extra$computeSeasonAmp`: logical (default to FALSE). If TRUE, compute and output the time-varying amplitude of the seasonality.

• `extra$computeTrendSlope`: logical (default to FALSE). If TRUE, compute and output the time-varying slope of the estimated trend.

• `extra$tallyPosNegSeasonJump`: logical (default to FALSE). If TRUE, compute and differentiate seasonal changepoints in terms of the direction of the jumps in the estimated seasonal signal. Those changepoints with a positive jump will be outputted separately from those with a negative jump. A series of output variables (some for positive-jump changepoints, and others for negative-jump changepoints will be dumped).

• `extra$tallyPosNegTrendJump`: logical (default to FALSE). If TRUE, compute and differentiate trend changepoints in terms of the direction of the jumps in the estimated trend. Those changepoints with a positive jump will be outputted separately from those with a negative jump. A series of output variables (some for positive-jump changepoints, and others for negative-jump changepoints will be dumped).

• `extra$tallyIncDecTrendJump`: logical (default to FALSE). If TRUE, compute and differentiate trend changepoints in terms of the direction of the jumps in the estimated trend. Those changepoints with an increase in the slope will be outputted separately from those with a decrease in the slope. A series of output variables (some for increase-jump changepoints, and others for decrease-jump changepoints will be dumped).

• `extra$printProgressBar`: logical (default to FALSE). If TRUE, a progress bar will be displayed to show the status of the running. When running on multiple time series (e.g. stacked image time series), the progress bar will also report an estimate of the remaining time for completion.

• `extra$consoleWidth`: integer (default to 0); the length of chars in each status line when setting `printProgressBar=TRUE`. If 0, the current width of the console will be used.
• `extra$printOptions`: logical (default to FALSE). If TRUE, the values used in the arguments `metadata`, `prior`, `mcmc`, and `extra` will be printed to the console at the start of the run.
• `extra$numThreadsPerCPU`: integer (default to 2); the number of threads to be scheduled for each CPU core.
• `extra$numParThreads`: integer (default to 0). When handling many time series, BEAST can use multiple concurrent threads. `extra$numParThreads` specifies how many concurrent threads will be used in total. If `numParThreads=0`, the actual number of threads will be `numThreadsPerCPU * cpuCoreNumber`; that is, each CPU core will generate a number `numThreadsPerCPU` of threads. On Windows 64, BEAST is group-aware and will affinely distribute the threads to all the NUMA node. But currently, up to 256 CPU cores are supported.

season
characters (default to 'harmonic'); specify if y has a periodic component or not. Three strings are possible.
• 'none': y is trend-only; no periodic components are present in the time series. The args for the seasonal component (i.e., `sorder.minmax`, `scp.minmax` and `sseg.max`) will be irrelevant and ignored.
• 'harmonic': y has a periodic/seasonal component. The term `season` is a misnomer, being used here to broadly refer to any periodic variations present in y. The periodicity is NOT a model parameter estimated by BEAST but a known constant given by the user through `freq`. By default, the periodic component is modeled as a harmonic curve—a combination of sins and cosines.
• If 'dummy', the same as 'harmonic' except that the periodic/seasonal component is modeled as a non-parametric curve. The harmonic order arg `sorder.minmax` is irrelevant and is ignored.

Value
The output is an object of class "beast". It is a list, consisting of the following variables. Exact sizes of the variables depend on the types of the input Y as well as the specified output time dimension `extra$whichOutputDimIsTime`. In the explanations below, we assume the input Y is a single time series of length N; the dimensions for 2D or 2D inputs may be interpreted accordingly:

`time` a vector of size 1xN: the times at the N sampled locations. By default, it is simply set to 1:N
`data` a vector, matrix, or 3D array; this is a copy of the input Y if `extra$dumpInputData` = TRUE. If `extra$dumpInputData`=FALSE, it is set to NULL. If the original input Y is irregular, the copy here is the regular version aggregated from the original at the time interval specified by `metadata$deltaTime`.
`marg_lik` numeric; the average of the model marginal likelihood; the larger marg_lik, the better the fitting for a given time series.
`R2` numeric; the R-square of the model fitting.
`RMSE` numeric; the RMSE of the model fitting.
sig2 numeric; the estimated variance of the model error.

trend a list object numeric consisting of various outputs related to the estimated trend component:

- *ncp*: [Number of ChangePoints]. a numeric scalar; the mean number of trend changepoints. Individual models sampled by BEAST has a varying dimension (e.g., number of changepoints or knots), so several alternative statistics (e.g., *ncp_mode*, *ncp_median*, and *ncp_pct90*) are also given to summarize the number of changepoints. For example, if `mcmc$samples=10`, the numbers of changepoints for the 10 sampled models are assumed to be c(0, 2, 4, 1, 1, 2, 7, 6, 6, 1). The mean ncp is 3.1 (rounded to 3), the median is 2.5 (2), the mode is 1, and the 90th percentile (*ncp_pct90*) is 6.5.

- *ncp_mode*: [Number of ChangePoints]. a numeric scalar; the mode for number of changepoints. See the above for explanations.

- *ncp_median*: [Number of ChangePoints]. a numeric scalar; the median for number of changepoints. See the above for explanations.

- *ncp_pct90*: [Number of ChangePoints]. a numeric scalar; the 90th percentile for number of changepoints. See the above for explanations.

- *ncpPr*: [Probability of the Number of ChangePoints]. A vector of length `prior$trendMaxKnotNum+1`. It gives a probability distribution of having a certain number of trend changepoints over the range of `[0,prior$trendMaxKnotNum]`; for example, `ncpPr[1]` is the probability of having no trend changepoint; `ncpPr[i]` is the probability of having (i-1) changepoints: Note that it is `ncpPr[i]` not `ncpPr[i-1]` because `ncpPr[1]` is used for having zero changepoint.

- *cpOccPr*: [ChangePoint OCCurence PRobability]. a vector of length N; it gives a probability distribution of having a changepoint in the trend at each point of time. Plotting `cpOccPr` will depict a continous curve of probability-of-being-changepoint. Of particular note, in the curve, a higher peak indicates a higher chance of being a changepoint only at that particular SINGLE point in time and does not necessarily mean a higher chance of observing a changepoint AROUND that time. For example, a window of `cpOccPr` values c(0, 0, 0.5, 0, 0) (i.e., the peak prob is 0.5 and the summed prob is 0.5) is less likely to be a changepoint compared to another window c(0.1, 0.2, 0.21, 0.2, 0.1) (i.e., the peak prob is 0.21 but the summed prob is 0.71).

- *order*: a vector of length N; the average polynomial order needed to approximate the fitted trend. As an average over many sampled individual piece-wise polynomial trends, order is not necessarily an integer.

- *cp*: [Changepoints] a vector of length `tcp.max=tcp.minmax[2]`; the most possible changepoint locations in the trend component. The locations are obtained by first applying a sum-filtering to the `cpOccPr` curve with a filter window size of `tseg.min` and then picking up to a total `prior$MaxKnotNum/tcp.max` of the highest peaks in the filtered curve. NaNs are possible if no enough changepoints are identified. `cp` records all the possible changepoints identified and many of them are bound to be false positives. Do not blindly treat all of them as actual changepoints.
• cpPr: [Changepoints PRobability] a vector of length metadata$trendMaxKnotNum; the probabilities associated with the changepoints cp. Filled with NaNs for the remaining elements if ncp<trendMaxKnotNum.

• cpCI: [Changepoints Credible Interval] a matrix of dimension metadata$trendMaxKnotNum x 2; the credible intervals for the detected changepoints cp.

• cpAbruptChange: [Abrupt change at Changepoints] a vector of length metadata$trendMaxKnotNum; the jumps in the fitted trend curves at the detected changepoints cp.

• Y: a vector of length N; the estimated trend component. It is the Bayesian model averaging of all the individual sampled trend.

• SD: [Standard Deviation] a vector of length N; the estimated standard deviation of the estimated trend component.

• CI: [Standard Deviation] a matrix of dimension N x 2; the estimated credible interval of the estimated trend. One vector of the matrix is for the upper envelope and another for the lower envelope.

• slp: [Slope] a vector of length N; the time-varying slope of the fitted trend component.

• slpSD: [Standard Deviation of Slope] a vector of length N; the SD of the slope for the trend component.

• slpSgnPosPr: [Probability of slope having a positive sign] a vector of length N; the probability of the slope being positive (i.e., increasing trend) for the trend component. For example, if slpSgnPosPr=0.80 at a given point in time, it means that 80% of the individual trend models sampled in the MCMC chain has a positive slope at that point.

• slpSgnZeroPr: [Probability of slope being zero] a vector of length N; the probability of the slope being zero (i.e., a flat constant line) for the trend component. For example, if slpSgnZeroPr=0.10 at a given point in time, it means that 10% of the individual trend models sampled in the MCMC chain has a zero slope at that point. The probability of slope being negative can be obtained from 1-slpSgnZeroPr-slpSgnPosPr.

• pos_ncp:
  • neg_ncp:
  • pos_ncpPr:
  • neg_ncpPr:
  • pos_cpOccPr:
  • neg_cpOccPr:
  • pos_cp:
  • neg_cp:
  • pos_cpPr:
  • neg_cpPr:
  • pos_cpAbruptChange:
  • neg_cpAbruptChange:
  • pos_cpCI:
  • neg_cpCI: The above variables have the same outputs as those variables without the prefix 'pos' and 'neg', except that we differentiate the changepoints with a POSitive jump in the trend from those changepoints with a
NEGative jump. For example, pos_ncp refers to the average number of trend changepoints that jump up (i.e., positively) in the trend.

- inc_ncp:
- dec_ncp:
- inc_ncpPr:
- dec_ncpPr:
- inc_cp0occPr:
- dec_cp0occPr:
- inc_cp:
- dec_cp:
- inc_cpPr:
- dec_cpPr:
- inc_cpAbruptChange:
- dec_cpAbruptChange:
- inc_cpCI:
- dec_cpCI: The above variables have the same outputs as those variables without the prefix ‘inc’ and ‘dec’, except that we differentiate the changepoints at which the trend slope increases from those changepoints at which the trend slope decreases. For example, if the trend slopes before and after a chngpt is 0.4 and 2.5, then the changepoint is counted toward inc_ncp.

season a list object numeric consisting of various outputs related to the estimated seasonal/periodic component:

- ncp: [Number of ChangePoints]. a numeric scalar; the mean number of seasonal changepoints.
- ncpPr: [Probability of the Number of ChangePoints]. A vector of length \( \text{prior$seasonMaxKnotNum+1} \). It gives a probability distribution of having a certain number of seasonal changepoints over the range of \([0, \text{prior$seasonMaxKnotNum}]\); for example, ncpPr[1] is the probability of having no seasonal changepoint; ncpPr[i] is the probability of having \((i-1)\) changepoints: Note that the index is \(i\) rather than \((i-1)\) because ncpPr[1] is used for having zero changepoint.
- cpOccPr: [ChangePoint OCCurrence PRObability]. a vector of length \(N\); it gives a probability distribution of having a changepoint in the seasonal component at each point of time. Plotting cpOccPr will depict a continuous curve of probability-of-being-changepoint over the time. Of particular note, in the curve, a higher value at a peak indicates a higher chance of being a changepoint only at that particular SINGLE point in time, and does not necessarily mean a higher chance of observing a changepoint AROUND that time. For example, a window of cpOccPr values \(c(0,0,0.5,0,0)\) (i.e., the peak prob is 0.5 and the summed prob is 0.5) is less likely to be a changepoint compared to another window values \(c(0.1,0.2,0.3,0.2,0.1)\) (i.e., the peak prob is 0.3 but the summed prob is 0.8).
- order: a vector of length \(N\); the average harmonic order needed to approximate the seasonal component. As an average over many sampled individual piece-wise harmonic curves, order is not necessarily an integer.
• cp: [Changepoints] a vector of length metadata$seasonMaxKnotNum; the most possible changepoint locations in the seasonal component. The locations are obtained by first applying a sum-filtering to the cpOccPr curve with a filter window size of prior$trendMinSeptDist and then picking up to a total ncp of the highest peaks in the filtered curve. If ncp<seasonMaxKnotNum, the remaining of the vector is filled with NaNs.

• cpPr: [Changepoints PRobability] a vector of length metadata$seasonMaxKnotNum; the probabilities associated with the changepoints cp. Filled with NaNs for the remaining elements if ncp<seasonMaxKnotNum.

• cpCI: [Changepoints Credible Interval] a matrix of dimension metadata$seasonMaxKnotNum x 2; the credible intervals for the detected changepoints cp.

• cpAbruptChange: [Abrupt change at Changepoints] a vector of length metadata$seasonMaxKnotNum; the jumps in the fitted trend curves at the detected changepoints cp.

• Y: a vector of length N; the estimated trend component. It is the Bayesian model averaging of all the individual sampled trend.

• SD: [Standard Deviation] a vector of length N; the estimated standard deviation of the estimated trend component.

• CI: [Standard Deviation] a matrix of dimension N x 2; the estimated credible interval of the estimated trend. One vector of the matrix is for the upper envelope and another for the lower envelope.

• amp: [AMPlitude] a vector of length N; the time-varying amplitude of the estimated seasonality.

• ampSD: [Standar Deviation of AMPlitude] a vector of length N; , the SD of the amplitude of the seasonality.

• pos_ncp:
• neg_ncp:
• pos_ncpPr:
• neg_ncpPr:
• pos_cpOccPr:
• neg_cpOccPr:
• pos_cp:
• neg_cp:
• pos_cpPr:
• neg_cpPr:
• pos_cpAbruptChange:
• neg_cpAbruptChange:
• pos_cpCI:
• neg_cpCI: The above variables have the same outputs as those variables without the prefix 'pos' and 'neg', except that we differentiate the changepoints with a POSitive jump in the trend from those changepoints with a NEGative jump. For example, pos_ncp refers to the average number of trend changepoints that jump up (i.e., positively) in the trend.

References


See Also

beast, beast.irreg, minesweeper, tetris, geeLandsat

Examples

```
# beast123() is an all-inclusive function that duplicates the functionalities of beast # and beast.irreg. It can handle a single, multiple, or 3D of stacked time series, being # either regular or irregular. It allows for customization through four LIST arguments: # metadata -- additional info about the input Y # prior -- prior parameters for the beast model # mcmc -- MCMC simulation setting # extra -- misc parameters turning on/off outputs and setting up parallel computations # Despite being essentially the same as beast and beast.irreg, beast123 is provided mainly to # support concurrent handling of multiple time series (e.g., stacked satellite images) via # parallel computing: When processing stacked raster layers, do not iterate pixels using # beast or beast.irreg via an external parallel caller (e.g., doParallel or foreach). Instead # use beast123, which supports mulithreading internally.

# Yellowstone is a half-monthly time series of 774 NDVI measurements at a Yellowstone # site starting from July 1-15,1981(i.e., start=c(1981,7,7)). It has 24 data points per # year (freq=24).

library(Rbeast)
data(Yellowstone)
plot(Yellowstone)
```

```
# The four option args are missing, so defalut values will be used, with some warning # messages given to altert this. By default, the input Y is assumed to be regular with # a seasonal component. The default values used will be printed out and they can serve # as a template to customize the parameters.

o = beast123(Yellowstone)
plot(o)
```
# Nile is an annual river flow time series (i.e., no periodic variation). So, season
# is set to 'none' for trend-only analysis. Default values are used for other
# missing options. Unlike the beast function, beast123 does NOT use the time attributes
# of a 'ts' object. For example, Nile is treated as a pure data number; its (start=1871,
# end=1970, freq=1) attributes are ignored. The default times 1:length(Nile) are used
# instead. The true time info need to be specified by the metadata parameter, as shown
# in the next example.

```r
o = beast123(Nile, season='none')
plot(o)
```

# Specify metadata, prior, mcmc, and extra explicitly. Only 'prior' is the true model
# parameters of BEAST; the other three are just options to configure the input/output or
# the computation process.

```r
# metadata is NOT part of BEAST itself, but some extra info to describe the input
# time series Y. Below, the input Y is the 'Yellowstone' ts.
metadata = list()
metadata$IsRegularOrdered = TRUE
metadata$whichDimIsTime = 1
metadata$startTime = c(1981,7,7) # Or startTime=1981.5137
metadata$deltaTime = 1/24 # Half-monthly regular ts: 0.5/12=1/24
metadata$period = 1.0 # The period is 1 year:
# freq x deltaTime = period
# 24 x 1/24 = 1.0
metadata$omissionValue = NaN
metadata$maxMissingRateAllowed = 0.7500 # If missingness is higher than .75, the ts
# is skipped and not fitted
metadata$deseasonalize = FALSE # Do not remove the global seasonal pattern
# before fitting the beast model
metadata$detrend = FALSE # Do not remove the global trend before
# the fitting

# prior is the ONLY true parameters of the beast model, used to specify the priors
# in the Bayesian formulation
prior = list()
prior$seasonMinOrder = 1 # min harmonic order allowed to fit seasonal cmpnt
prior$seasonMaxOrder = 5 # max harmonic order allowed to fit seasonal cmpnt
prior$seasonMinKnotNum = 0 # min number of changepnts in seasonal cmpnt
prior$seasonMaxKnotNum = 3 # max number of changepnts in seasonal cmpnt
prior$seasonMinSepDist = 10 # min inter-chngpts separation for seasonal cmpnt
prior$trendMinOrder = 0 # min polynomial order allowed to fit trend cmpnt
prior$trendMaxOrder = 1 # max polynomial order allowed to fit trend cmpnt
prior$trendMinKnotNum = 0 # min number of changepnts in trend cmpnt
prior$trendMaxKnotNum = 15 # max number of changepnts in trend cmpnt
```
prior$trendMinSepDist = 5  # min inter-chngpts separation for trend cmpnt
prior$precValue = 10.0  # Initial value of the precision parameter (no
# need to change it unless for precPrioType='const')
prior$precPriorType = 'uniform'  # Possible values: const, uniform, and componentwise

# mcmc is NOT part of the beast model itself, but some parameters to configure the
# MCMC inference.
mcmc = list()

mcmc$seed = 9543434  # an arbitray seed for random number generator
mcmc$samples = 3000  # samples collected per chain
mcmc$thinningFactor = 3  # take every 3rd sample and discard others
mcmc$burnin = 150  # discard the initial 150 samples per chain
mcmc$chainNumber = 3  # number of chains
mcmc$maxMoveStepSize = 4  # max random jump step when proposing new chngpts
mcmc$trendResamplingOrderProb = 0.100  # prob of choosing to resample polynomial order
mcmc$seasonResamplingOrderProb = 0.100  # prob of choosing to resample harmonic order
mcmc$credIntervalAlphaLevel = 0.950  # the significance level for credible interval

# extra is NOT part of the beast model itself, but some parameters to configure the
# output and computation process
extra = list()
extra$dumpInputData = FALSE  # If true, a copy of input time series is outputted
extra$whichOutputDimIsTime = 1  # For 2D or 3D inputs, which dim of the output refers to
# time? Ignored if the input is a single time series
extra$computeCredible = FALSE  # If true, compute CI: computing CI is time-intensive.
extra$fastCIComputation = TRUE  # If true, a faster way is used to get CI, but it is
# still time-intensive. That is why the function beast()  
# is slow because it always compute CI.
extra$computeSeasonOrder = FALSE  # If true, dump the estimated harmonic order over time
extra$computeTrendOrder = FALSE  # If true, dump the estimated polynomial order over time
extra$computeSeasonChngpt = TRUE  # If true, get the most likely locations of s chngpts
extra$computeTrendChngpt = TRUE  # If true, get the most likely locations of t chngpts
extra$computeSeasonAmp = FALSE  # If true, get time-varying amplitude of seasonality
extra$computeTrendSlope = FALSE  # If true, get time-varying slope of trend
extra$tallyPosNegSeasonJump = FALSE  # If true, get those changpts with +/- jumps in season
extra$tallyPosNegTrendJump = FALSE  # If true, get those changpts with +/- jumps in trend
extra$tallyIncDecTrendJump = FALSE  # If true, get those changpts with increasing/
# decreasing trend slopes
extra$printProgressBar = TRUE
extra$printOptions = TRUE
extra$consoleWidth = 0  # If 0, the console width is from the current console
extra$numThreadsPerCPU = 2  # 'numThreadsPerCPU' and 'numParThreads' are used to
extra$numParThreads = 0  # configure multithreading runs; they're used only if
# Y has multiple time series (e.g., stacked images)

o = beast123(Yellowstone,metadata,prior,mcmc,extra, season='harmonic')
plot(o)

## End(Not run)
# Handle irregular time series: ohio is a data frame of a Landsat NDVI series observed at unevenly-spaced times

data(ohio)
str(ohio)

metadata = list()
metadata$isRegularOrdered = FALSE  # The input data is irregular
metadata$time = ohio$time # Must supply individual times for irregular inputs
metadata$deltaTime = 1/12 # Must supply the desired time interval for aggregation
metadata$period = 1.0

o=beast123(ohio$ndvi, metadata)  # Default values used for those missing parameters

# Another accepted time format for beast123

metadata = list()
metadata$time = ohio$rdate # Must supply individual times for irregular inputs

o=beast123(ohio$ndvi, metadata)  # Default values used for those missing parameters

# Another accepted time format for beast123

metadata = list()
metadata$time$year = ohio$Y
metadata$time$month = ohio$M
metadata$time$day = ohio$D

o=beast123(ohio$ndvi, metadata)  # Default values used for those missing parameters

# Another accepted time format for beast123

metadata = list()
metadata$time$year = ohio$Y
metadata$time$doy = ohio$doy

o=beast123(ohio$ndvi, metadata)  # Default values used for those missing parameters

# Another accepted time format for beast123

metadata = list()
metadata$time = ohio$time

o=beast123(ohio$ndvi, metadata)  # Default values used for those missing parameters

# Another accepted time format for beast123

metadata = list()
metadata$isRegularOrdered = FALSE # Irregular input
metadata$deltaTime = 1/12 # Must supply the desired time interval for aggregation
metadata$time = ohio$time # Fractional year

o=beast123(ohio$ndvi, metadata) # Default values used for those missing parameters

-------------------------------------------------------------------------------------------------------

ohio$datestr1 # Another accepted time format for beast123

metadata = list()
metadata$isRegularOrdered = FALSE # Irregular input
metadata$deltaTime = 1/12 # Must supply the desired time interval for aggregation
metadata$time$datestr = ohio$datestr1
metadata$time$strfmt = '????yyyy?mm?dd'
o=beast123(ohio$ndvi, metadata) # Default values used for those missing parameters

-------------------------------------------------------------------------------------------------------

ohio$datestr2 # Another accepted time format for beast123

metadata = list()
metadata$isRegularOrdered = FALSE # Irregular input
metadata$deltaTime = 1/12 # Must supply the desired time interval for aggregation
metadata$time$datestr = ohio$datestr2
metadata$time$strfmt = '????yyyydoy???'
o=beast123(ohio$ndvi, metadata) # Default values used for those missing parameters

-------------------------------------------------------------------------------------------------------

ohio$datestr3 # Another accepted time format for beast123

metadata = list()
metadata$isRegularOrdered = FALSE # Irregular input
metadata$deltaTime = 1/12 # Must supply the desired time interval for aggregation
metadata$time$datestr = ohio$datestr3
metadata$time$strfmt = 'Y,,M/D'
o=beast123(ohio$ndvi, metadata) # Default values used for those missing parameters

-------------------------------------------------------------------------------------------------------

# Handle multiple time series: 'simdata' is a 2D matrix of dim 300x3; it consists of 3
time series of length 300 each. As a toy example, the 3 time series are the same.
## Not run:
data(simdata) #
dim(simdata) # dim of simdata: 300 x 3 (time x num_of_ts)

metadata = list() # Which dim of the input refer to time for 2D inputs?
metadata$whichDimIsTime = 1 # 300 is the ts length, so dim is set to '1' here.
metadata$period = 24 # By default, we assume startTime=1 and deltaTime=1

extra=list()
extra$whichOutputDimIsTime = 2 # Which dim of the output arrays refers to time?
o=beast123(simdata, metadata,extra=extra) # Default values used for those missing parameters
Another run by transposing simdata

```r
simdata1 = t(simdata)  # dim of simdata1: 3 x 300 (num of ts x time)
```

```r
metadata = list()
metadata$whichDimIsTime = 2  # Which dim of the input refer to time for 2D inputs?
# 300 is the ts length, so dim is set to '2' here.
metadata$period = 24  # By default, we assume startTime=1 and deltaT=1
```

```r
o = beast123(simdata1, metadata)  # Default values used for those missing parameters
```

## End(Not run)

---

Handle 3D stacked images of irregular and unordered time-series: imagestack is a 3D
# array of size 12x9x1066, each pixel being a time series of length 1066

```r
imagestack
data(imagestack)
dim(imagestack$ndvi)  # Dim: 12 x 9 x 1066 (row x col x time)
imagestack$datestr  # A character vector of 1066 date strings
```

```r
metadata = list()
metadata$isValidInput = FALSE  # 'imagestack$ndvi' is an IRREGULAR input
metadata$whichDimIsTime = 3  # Which dim of the input refer to time for 3D inputs?
# 1066 is the ts length, so dim is set to '3' here.
metadata$startTime$datestr = imagestack$datestr
metadata$startTime$strfmt = 'LT05_018032_20080311.yyyyMMdd'
metadata$deltaTime = 1/12  # Aggregate the irregular ts at a monthly interval: 1/12 Yr
metadata$period = 1.0  # The period is 1 year: deltaTime*freq=1/12*12=1.0
```

```r
extra = list()
extra$dumpInputData = TRUE  # Get a copy of aggregated input ts
extra$numThreadsPerCPU = 2  # Each cpu core will be assigned 2 threads
extra$numParThreads = 0  # If 0, total_num_threads=numThreadsPerCPU*num_of_cpu_core
# if >0, used to specify the total number of threads
```

```r
# Default values for missing parameters
o = beast123(imagestack$ndvi, metadata=metadata, extra=extra)
```

```r
print(o,c(5,3))  # print the result for the pixel at Row 5 and Col 3
plot(o,c(5,3))  # plot the result for the pixel at Row 5 and Col 3
image(o$trend$ncp)  # number of trend changepoints over space
```

## End(Not run)
CNAchrom11

DNA copy number alteration data in array-based CGH data for Chromosome 11

Description

CNAchrom11 is a vector of the log2 intensity ratios for cell line GM03576 for Chromosome 11, obtained from Snijders et al. (2001).

Usage

data(CNAchrom11)

Source

Snijders et al. (2001), Assembly of microarrays for genome-wide measurement of DNA copy number, Nature Genetics, 29, 263-264 (http://www.nature.com/ng/journal/v29/n3/full/ng754.html).

References


Examples

library(Rbeast)
data(CNAchrom11)

o = beast(CNAchrom11, season='none') # no periodic component
plot(o)
# Description

covid19 is a data frame consisting of daily confirmed COVID19 cases and deaths in the world from Jan 22, 2020 to Dec 16, 2021.

## Usage

data(covid19)

## Source

https://ourworldindata.org/grapher/daily-covid-cases-deaths?country=~OWID_WRL (last accessed on Dec 16, 2021)

## References


## Examples

library(Rbeast)
data(covid19)

## Not run:
newcases = covid19$newcases

# This time series has a periodical variation of 7 days. 7 days can't be precisely represented in the unit of year bcz some years has 365 days and others has 366.
# So, here we use the date number as the time unit (i.e., the number of days lapsed since 1970-01-01).
datenum = as.numeric(covid19$date)
o = beast(newcases, start=min(datenum), deltat=1, freq=7)
geeLandsat

Landsat reflectance and NDVI time series from Google Earth Engine

Description

Get Landsat reflectance and NDVI time series from Google Earth Engine given longitude and latitude

Usage

geeLandsat(lon=NA, lat=NA, radius=100, stat='mean', timeout=700)

Arguments

lon numeric within [-180,180]
lat numeric within [-90,90]
radius a positive number ( <=500 meters ); the radius of a buffer around the given latitude and longitude for aggregation. If radius=0, the single pixel at the lat and lon will be retrieved
stat character; if radius>0, used to specify the spatial aggregation method for pixels in the buffer. Possible values are 'mean', 'min', 'max', or 'median'.
timeout integer; the seconds elapsed to wait for connection timeout. See the note for an explanation.

Value

a data.frame object consisting of dates, sensor type, reflectances, and NDVI for the requested location. It contains only valid and clear-sky values as obtained by referring to the standard clouds flags.
**Note**

As a poor man’s scheme to interact with Google Earth Engine, geeLandsat should be used only for occasional retrieval of Landsat time series at a few sites, NOT for batch downloading for thousands of sites in a R loop. This procedure is provided to get example time series for testing BEAST. Behind the scene, this function calls to a free Python-based server using my own GEE credential. Normally it takes several seconds to retrieve one time series, but as a free cloud service, the Python server only offers 100 seconds of free CPU time per day, with throttling applied. So it may take up to a few mins to get a time series on your end. It may fail due to connection timeout; if so, give it a few tries. If you need to retrieve data for thousands or millions of sites, please contact the author.

**References**


**See Also**

`beast, beast.irreg, beast123, minesweeper, tetris`

**Examples**

```r
library(Rbeast)
## Not run:
df = geeLandsat(lon=-80.983877, lat = 40.476882) # if it fails, try a few more times before giving up
print(df)
## End(Not run)
```

---

A monthly Google Trend time series of the US search interest in the word "beach"
**Description**

googletrend_beach is a ts object comprising monthly search interest in "beach" from the United States, as reported from Google Trends. Sudden changes in the search trend are attributed to extreme weather events or the covid19 outbreak.

**Usage**

data(googletrend_beach)

**Source**

https://trends.google.com/trends/explore?date=all&geo=US&q=beach

**References**


**Examples**

```r
library(Rbeast)
data(googletrend_beach) # A monthly ts starting from Jan 2004

o = beast(googletrend_beach)
plot(o)
```

---

**Description**

imagestack is a LIST containing Landsat-derived NDVI image chips at an Ohio site.

**Usage**

data(imagestack)
Source

Landsat images courtesy of the U.S. Geological Survey

References


Examples

data(imagestack)
imagestack$datestr # A string vector containg the observation dates of individual ndvi images
## Not run:
declare(imagestack$ndvi) # NDVI images collected over the past several decades
## End(Not run)
plot(imagestack$ndvi[3,4,],type='l') # Plot the raw data at a pixel

---

**minesweeper**

The Minesweeper game in R

Description

A poor man’s implementation of the minesweeper game in R. Yes, you are right: it has nothing to do with time series decomposition, changepoint detection, and time series segmentation. Its only remote connection to Rbeast is that this is a practice script I wrote to learn R graphics for implementing Rbeast.

Usage

minesweeper(height=15, width=12, prob=0.1)
**minesweeper**

**Arguments**

- `height` integer; number of rows of the mine grid along the vertical direction.
- `width` integer; number of columns of the mine grid along the horizontal direction.
- `prob` numeric; a fraction between 0 and 1 to specify the probability of mine occurrence in the mine grid.

**Value**

Instructions:

- LEFT-click to clear a spot.
- RIGHT-click to flag a spot.
- MIDDLE-click(wheel) a cleared and numbered spot to open neighbor spots, if flagged correctly.
- Click Restart for a new game

**Note**

An interactive graphics window is needed to run this function correctly. So it won’t run in RStudio’s plot pane. The function will use the x11() or x11(type='Xlib') graphic device to open a pop-up window.

**References**


**See Also**

beast, beast.irreg, beast123, tetris, geeLandsat

**Examples**

```r
library(Rbeast)

## Not run:
```

ohio

An irregular Landsat NDVI time series at an Ohio site

Description

ohio is a data.frame object comprising decades of Landsat-observed surface reflectances and NDVI at an Ohio site

Usage

data(ohio)

Source

Landsat images courtesy of the U.S. Geological Survey

References


Examples

library(Rbeast)
data(ohio) # Landsat surface references and NDVI at a single pixel observed over time
str(ohio)

## Not run:
# ohio$ndvi is a single irregular time series
```r
y = ohio$ndvi
o = beast.irreg(y, time=ohio$time, deltat=1/12)
plot(o)
print(o)

# Ohio also contains irregular time series of individual spectral bands
# Below, run the multivariate version of the BEAST algorithm to decompose
# the 5 time series and detect common changepoints altogether

y = list(ohio$blue, ohio$green, ohio$red, ohio$nir, ohio$swir1);
o = beast.irreg(y, time=ohio$time, deltat=1/12, freq=12)
plot(o)
print(o)

## End (Not run)
```

---

**plot.beast**  

_Bayesian changepoint detection and time series decomposition_

**Description**

Plot the result obtained from the `beast` function.

**Usage**

```r
## S3 method for class 'beast'
plot(
x,
index = 1,
vars = c('st', 's', 'scp', 'sorder', 't', 'tcp', 'torder', 'slpsgn', 'o', 'ocp', 'error'),
col = NULL,
main = "BEAST decomposition and changepoint detection",
xlab = 'Time',
ylab = NULL,
cex.main = 1,
cex.lab = 1,
relative.heights = NULL,
interactive = FALSE,
ncpStat = c('median', 'mode', 'mean', 'pct90', 'max'),
...)
```

**Arguments**

- `x`  
  a "beast" object returned by `beast.beast.irreg` or `beast123`. It may contain one or many time series.
index an integer (default to 1) or a vector of two integers to specify the index of the
time series to plot if x contains results for multiple time series. index is always 1
if x has 1 time series. If x is returned by beast123 with a 2D input, index should
be a single integer. If x is from beast123 applied to 3D arrays of time series
(e.g., stacked satellite images), index can be a linear index or two subscripts to
specify the row and column of the pixel/grid.

vars a vector of strings indicating the elements or variables of x to plot. Possible
vars strings include 'st' (season plus trend), 's' (season component), 't' (trend
component), 'o' (outliers), 'scp', 'tcp', 'ocp' (occurrence probability of sea-
onal/trend/outlier changepoint), 'sorder' (seasonal harmonic order), 'torder'
(trend polynomial order), 'samp' (amplitude of seasonality), 'tslp' (slope of
trend), 'slpsgn' (probabilities of the slope being positive, zero, and negative)
and 'error' (remainder).

relative.heights a numeric vector of the same length as that of vars to specify the relative heights
of subplots of individual variables in vars.

col a string vector of the same length as that of vars to specify the colors of indi-
vidual subplots associated with vars.

main a string; the main title.

xlab a string: the x axis title.

ylab a string vector of the same length as that of vars to specify the y axis names of
individual subplots associated with vars

cex.main cex for the main title

cex.lab cex for the axis title

interactive a bool scalar. If TRUE, an interactive GUI is used for examining individual
elements of x.

ncpStat character. A string to specify which statistic is used for the Number of Change-
Point (ncp). Five values are possible: 'mean', 'mode', 'median', 'pct90', and
'max'; the default is 'median'. Individual models sampled by BEAST has a
varying dimension (e.g., number of changepoints or knots). For example, if
mcmc$samples=10, the numbers of changepoints for the 10 sampled models are
assumed to be c(0, 2, 4, 1, 1, 2, 7, 6, 6, 1). The mean ncp will be 3.1 (rounded to
3), the median is 2.5 (2), the mode is 1, and the maximum is 7. The 'max' op-

tion plots all the changepoints recorded in out$trend$cp, out$season$cp, or
out$outlier$cp; many of these changepoints are bound to be false positives,
so do not treat all of them as actual changepoints.

Value

This function creates various plots to demonstrate the results of a beast decomposition.

References

1. Zhao, K., Wulder, M.A., Hu, T., Bright, R., Wu, Q., Qin, H., Li, Y., Toman, E., Mallick,
B., Zhang, X. and Brown, M., 2019. Detecting change-point, trend, and seasonality in satel-
lite time series data to track abrupt changes and nonlinear dynamics: A Bayesian ensemble

sensing of plant biochemistry using Bayesian model averaging with variable and band selec-
tion. Remote Sensing of Environment, 132, pp.102-119 (the Bayesian MCMC scheme used
in beast).

Mapping fine-scale human disturbances in a working landscape with Landsat time series on
Google Earth Engine. ISPRS Journal of Photogrammetry and Remote Sensing, 176, pp.250-
261 (a beast application paper).

See Also
    beast, beast.irreg, beast123, plot.beast, minesweeper, tetris, geeLandsat

Examples

library(Rbeast)
data(simdata)
## Not run:
result=beast123(simdata, metadata=list(whichDimIsTime=1))
plot(result,1)
plot(result,2)
## End(Not run)

print.beast  

Bayesian changepoint detection and time series decomposition

Description

Summarize and print the results obtained from the BEAST time series decomposition and segmen-
tation.

Usage

## S3 method for class 'beast'
print(
  x,
  index = 1,
  ...
)
Arguments

- **x**: a "beast" object returned by `beast`, `beast.irreg`, or `beast123`. It may contain one or many time series.

- **index**: an integer (default to 1) or a vector of two integers to specify the index of the time series to print if `x` contains results for multiple time series. If `x` has 1 time series, `index` should be always 1. If `x` is returned by `beast123` applied to a 2D input, `index` should be a single index. If `x` is from `beast123` applied to 3D arrays of time series (e.g., stacked satellite images), `index` can be a linear index or two subscripts to specify the row and column of the desired pixel/grid.

- ... additional parameters to be implemented.

Value

Print a summary of changepoints detected for the seasonal or trend component.

References


See Also

`beast`, `beast.irreg`, `beast123`, `minesweeper`, `tetris`, `geeLandsat`

Examples

```r
library(Rbeast)

data(simdata)

## Not run:
# out=beast123(simdata) # Error: whichDimIsTime has to be specified. See below
out=beast123(simdata, metadata=list(whichDimIsTime=1))
print(out, 1)
print(out, 2)

## End(Not run)
```
Description

simdata is a 300 x 3 matrix, consisting three time series of length 300. Currently, the three time series are the same. It is used to illustrate BEAST can handle multiple time series at a single function call. of BEAST.

Usage

data(simdata)

Source

Rbeast v0.9.2

References


Examples

library(Rbeast)
data(simdata)
plot(simdata[,1],type='l')

## Not run:
#out=beast123(simdata) # Error: whichDimIsTime has to be specified. See below
out=beast123(simdata, metadata=list(whichDimIsTime=1))

plot(out,1)
plot(out,2)
plot(out,3)

## End(Not run)
Description

A poor man's implementation of the Tetris game in R. Yes, you are right again: it has nothing to do with time series decomposition, changepoint detection, and time series segmentation. Its only remote connection to Rbeast is that this is a practice script I wrote to learn R graphics for implementing Rbeast.

Usage

tetris(height=25, width=14, speed=0.6)

Arguments

- height: integer; number of rows of the mine grid along the vertical direction.
- width: integer; number of columns of the mine grid along the horizontal direction.
- speed: numeric; a time interval between 0.05 and 2 seconds, specifying how fast the tetrinimos moves down. The smaller, the faster.

Value

Instructions:
- Left arrow to move left.
- Right arrow to move right.
- Up arrow to rotate.
- Down arrow to speed up.
- Space key to sink to the bottom.

Note

This function works only under the Windows OS not Linux or Mac. An interactive graphics window is needed to run this function correctly. So it won’t run in RStudio’s plot pane. The function will use the x11() or x11(type='Xlib') graphic device to open a pop-up window.

References


See Also

beast, beast.irreg, beast123, minesweeper, geeLandsat

Examples

library(Rbeast)

## Not run:
tetris()

# A field of size 20x25 with blocks moving down every 0.1 sec.
tetris(20,25,0.1)

## End(Not run)

tsextract

Bayesian changepoint detection and time series decomposition

Description

Extract the result of a single time series from an object of class beast

Usage

tsextract( x, index = 1 )

Arguments

x

a "beast" object returned by beast, beast.irreg, or beast123. It may contain one or many time series.

index

an integer (default to 1 ) or a vector of two integers to specify the index of the time series to extract if x contains results for multiple time series. If x has 1 time series, index should be always 1. If x is returned by beast123 applied to a 2D input, index should be a single index. If x is from beast123 applied to 3D arrays of time series (e.g., stacked satellite images), index can be a linear index or two subscripts to specify the row and column of the desired pixel/grid.
Value

A LIST object of the result for the chosen time series, which contains the same field as x.

Note

Use this function only to manually and interactively examine individual times series. If the purpose is to loop through x, the use of direct indexing is much faster. For example, if x is a beast object for a 300x200x1000 3D array (row x col x time), use x$trend$Y[20,40,] to get the fitted trend at the pixel of row 20 and col 40.

References


See Also

beast, beast.irreg, beast123, minesweeper, tetris, geeLandsat

Examples

```r
library(Rbeast)
data(simdata)

# handle only the 1st ts
out=beast(simdata[,1])

## Not run:
# handle all the ts
out=beast123(simdata, metadata=list(whicDimIsTime=1))

plot(out,1)
plot(out,2)

## End(Not run)
```
**Description**

Yellowstone is a vector comprising 30 years’ AVHRR NDVI data at a Yellowstone site

**Usage**

```r
data(Yellowstone)
```

**Source**

Rbeast v0.9.2

**References**


**Examples**

```r
library(Rbeast)
data(Yellowstone)
plot(Yellowstone,type='l')

result=beast(Yellowstone)
plot(result)
```
Index

* misc
  beast, 2
  beast.irreg, 17
  beast123, 27
  CNAchrom11, 45
  covid19, 46
  geeLandsat, 47
  gooletrend_beach, 48
  imagestack, 49
  minesweeper, 50
  ohio, 52
  plot.beast, 53
  print.beast, 55
  simdata, 57
  tetris, 58
  tsextract, 59
  Yellowstone, 61

Ohio (ohio), 52
ohio, 3, 18, 52
plot.beast, 32, 53, 55
print.beast, 55
RBEAST (beast), 2
Rbeast (beast), 2
rbeast (beast), 2
simdata, 57
Tetris (tetris), 58
tetris, 11, 26, 39, 48, 51, 55, 56, 58, 60
tsextract, 59
Yellowstone, 61

BEAST (beast), 2
Beast (beast), 2
beast, 2, 11, 18, 26, 39, 48, 51, 53, 55, 56, 59, 60
beast.123 (beast123), 27
BEAST_irreg (beast.irreg), 17
beast.irreg, 3, 6, 7, 11, 17, 39, 48, 51, 53, 55, 56, 59, 60
BEAST123 (beast123), 27
beast123, 3, 6, 7, 11, 18, 26, 27, 48, 51, 53–56, 59, 60

CNAchrom11, 44
Covid19 (covid19), 46
covid19, 46

geeLandsat, 11, 26, 39, 47, 51, 55, 56, 59, 60
googletrend_beach, 48

imagestack, 49

minesweeper, 11, 26, 39, 48, 50, 55, 56, 59, 60